Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Diseño y Desarrollo de una Plataforma Web para la Gestión Interna de un Centro de Investigación

Autora: Detelina Gloria Valerieva Velcheva

Tutor: Antonio Jesús Díaz Honrubia

Madrid, enero 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Diseño y Desarrollo de una Plataforma Web para la Gestión Interna de un Centro de Investigación
Mes Año

Autor: Detelina Gloria Valerieva Velcheva

Tutor: Antonio Jesús Díaz Honrubia
LENGUAJES Y SISTEMAS INFORMÁTICOS E INGENIERÍA DE SOFTWARE
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

La gestión inadecuada de los recursos en un centro de investigación puede generar caos. La falta de una base de datos centralizada y de un sistema web para controlar estos recursos puede ocasionar la duplicación o pérdida de datos, lo que genera confusión, retrasa tareas críticas y provoca errores como la reserva simultánea de una misma aula o la modificación no autorizada de las vacaciones de otro investigador. Estos problemas afectan no solo a la organización interna, sino también a la productividad y eficiencia del centro, generando retrasos y errores en la planificación que impactan directamente en el rendimiento de los grupos de investigación.

Actualmente, el Centro de Tecnología Biomédica (CTB) carece de una infraestructura adecuada en términos de sistemas de gestión de información, lo que genera riesgos como la falta de trazabilidad de solicitudes, la manipulación de permisos y la posibilidad de que algunas personas hagan uso del sistema en beneficio propio. Un ejemplo de esto es que, en la confusión del sistema actual, se pueden aprovechar para obtener más días de vacaciones sin justificación. Sin un sistema eficiente, las tareas administrativas se complican innecesariamente, desvían recursos valiosos y dificultan la toma de decisiones informadas. Ante esta situación, resulta urgente implementar una plataforma centralizada que no solo resuelva estos problemas inmediatos, sino que también optimice la gestión de recursos y personal, garantizando la integridad, seguridad y trazabilidad de los datos.

Este proyecto comenzará con un análisis exhaustivo de las tecnologías disponibles, seguido de la identificación de los requisitos funcionales y no funcionales del sistema. A continuación, se desarrollará un prototipo inicial de alta fidelidad (Hi-Fi), que incluirá el diseño de la interfaz de usuario y las funcionalidades necesarias para satisfacer las necesidades del centro, comenzando con dos módulos clave: gestión de aulas y gestión de ausencias. Aunque la solución inicial se centrará en estos dos módulos, se diseñará de manera escalable, permitiendo la integración de otros recursos del centro en el futuro. Posteriormente, se diseñará la base de datos, priorizando eficiencia, seguridad y escalabilidad, y se definirá la arquitectura del sistema, aplicando principios de ingeniería de software, como la responsabilidad mínima e independencia de clases, para asegurar un código limpio, mantenible y escalable. La interfaz de usuario será diseñada de manera que sea intuitiva y accesible para los distintos perfiles de usuario dentro del centro, asegurando que todos puedan interactuar con el sistema sin dificultades. Tras la fase de desarrollo, se realizarán pruebas exhaustivas para garantizar que el sistema cumpla con los estándares de calidad y rendimiento.

La implementación de esta plataforma no solo mejorará la eficiencia operativa y reducirá los riesgos, sino que permitirá al personal centrarse en lo que realmente importa: la investigación. Con procesos administrativos optimizados y la mejora en la trazabilidad y transparencia, el CTB podrá maximizar su rendimiento, colaborando de manera más eficiente y contribuyendo significativamente al avance de la ciencia biomédica.

Abstract

The inadequate management of resources in a research center can lead to chaos. The lack of a centralized database and a web-based system to control these resources can result in data duplication or loss, causing confusion, delaying critical tasks, and leading to errors such as the simultaneous booking of the same classroom or unauthorized changes to another researcher's vacation time. These problems affect not only the internal organization but also the productivity and efficiency of the center, causing delays and errors in planning that directly impact the performance of research teams.

Currently, the Biomedical Technology Center (CTB) lacks adequate infrastructure in terms of information management systems, which creates risks such as the lack of traceability of requests, the manipulation of permissions, and the possibility of some individuals using the system for personal gain. An example of this is that, due to the confusion in the current system, people may take advantage of it to obtain more vacation days without justification. Without an efficient system, administrative tasks become unnecessarily complicated, divert valuable resources, and hinder informed decision-making. Given this situation, it is urgent to implement a centralized platform that not only addresses these immediate issues but also optimizes the management of resources and personnel, ensuring the integrity, security, and traceability of data.

This project will begin with a thorough analysis of the available technologies, followed by the identification of the system's functional and non-functional requirements. Next, a high-fidelity prototype (Hi-Fi) will be developed, including the design of the user interface and the necessary functionalities to meet the center's needs, starting with two key modules: room management and absence management. Although the initial solution will focus on these two modules, it will be designed to be scalable, allowing for the integration of other resources within the center in the future. Subsequently, the database will be designed with a focus on efficiency, security, and scalability, and the system architecture will be defined, applying software engineering principles such as minimal responsibility and class independence to ensure clean, maintainable, and scalable code. The user interface will be designed to be intuitive and accessible to the various user profiles within the center, ensuring that everyone can interact with the system with ease. After the development phase, extensive testing will be conducted to ensure the system meets quality and performance standards.

The implementation of this platform will not only improve operational efficiency and reduce risks but will also allow staff to focus on what truly matters: research. With optimized administrative processes and improved traceability and transparency, the CTB will be able to maximize its performance, collaborate more efficiently, and contribute significantly to the advancement of biomedical science.

Tabla de contenidos

Tabla de Figuras

Tabla de Tablas

1 Introducción

En este capítulo se presenta el proyecto, sus objetivos y la problemática que busca resolver. Se incluirá la planificación de tareas a través de un mapa de Gantt, así como una descripción de la estructura de la memoria para guiar al lector a lo largo del contenido y enfoque de los capítulos siguientes.

1.1 Motivación y necesidad del proyecto

En el ámbito de los centros de investigación, la gestión eficiente de recursos, como aulas y el control de horarios del personal, es crucial para optimizar las actividades diarias y garantizar un uso adecuado de las instalaciones. Sin embargo, en muchas instituciones, estas operaciones se siguen llevando a cabo de forma manual o mediante herramientas genéricas que no se adaptan completamente a las necesidades específicas del centro.

Esta situación puede desencadenar una serie de problemas que afectan negativamente la organización y el rendimiento del centro. Entre los más comunes se encuentran la duplicación de reservas de espacios, la falta de coordinación entre los departamentos, y la dificultad para gestionar las ausencias y vacaciones del personal. Estos inconvenientes no solo generan ineficiencias en la operativa diaria, sino que también pueden afectar la satisfacción del personal, dificultar la planificación estratégica y, en última instancia, limitar la capacidad del centro para cumplir con sus objetivos de investigación.

Frente a esta problemática, las soluciones digitales especializadas han demostrado ser herramientas clave para transformar y modernizar procesos administrativos en diversos sectores. Una plataforma web diseñada específicamente para la gestión interna de los centros de investigación representa una oportunidad única para abordar estas limitaciones.

Con una solución de este tipo, la información estará centralizada y siempre accesible para los usuarios autorizados, eliminando la incidencia de errores humanos relacionados con la duplicación de datos o la falta de actualizaciones en tiempo real. Además, se optimizará la asignación de recursos al garantizar que las aulas y otros espacios estén disponibles y reservados según las necesidades reales del personal y los proyectos en curso.

En definitiva, la implementación de una plataforma digital especializada no solo resolvería los problemas operativos actuales, sino que también permitiría establecer una gestión más ágil y eficiente a largo plazo. Al automatizar procesos clave, como la asignación de recursos y la gestión de horarios, se facilitaría la coordinación interna y se reducirían los errores. Además, al eliminar tareas burocráticas y repetitivas, el personal no perdería tiempo en gestiones administrativas, lo que les permitiría dedicar más tiempo a las actividades de investigación. Esto se traduciría en un aumento de la productividad y en una mejor utilización de los recursos disponibles. En definitiva, esta solución digital

ofrecería una forma más moderna y eficaz de gestionar las actividades del centro de investigación, alineándose con las necesidades de un entorno cada vez más interconectado y dinámico.

1.2 Objetivos

El presente Trabajo de Fin de Grado (TFG) tiene como objetivo principal el diseño y desarrollo de una plataforma web que permita gestionar de manera eficiente las reservas de aulas, así como el control de ausencias y vacaciones del personal en centros de investigación. Este proyecto busca ofrecer una solución digital integral que aborde las limitaciones actuales, proporcionando una herramienta escalable, eficiente y adaptada a las demandas específicas del entorno. Para alcanzar este objetivo principal, se definen los siguientes objetivos específicos:

1. **Estudio de tecnologías**: investigar y seleccionar herramientas tecnológicas que garanticen el rendimiento, la seguridad y la escalabilidad de la aplicación.
2. **Diseño de la plataforma**: establecer una arquitectura clara para el sistema, diseñando tanto la estructura de la base de datos como la lógica de la aplicación.
3. **Implementación de módulos funcionales:**
    a. **Gestión de salas**: crear un módulo para realizar reservas de aulas, evitando conflictos y permitiendo un control eficiente.
    b. **Gestión del personal**: desarrollar funcionalidades que faciliten la administración de ausencias y días de vacaciones, minimizando superposiciones y garantizando la transparencia.
4. **Seguridad e integración**: implementar mecanismos de cifrado para datos sensibles y un sistema de control de acceso basado en roles, asegurando la integridad y privacidad de la información.
5. **Testeo y validación**: ejecutar pruebas unitarias, integradas y funcionales para comprobar que la aplicación cumple con los requisitos establecidos y garantiza una experiencia de usuario satisfactoria.

1.3 Planificación

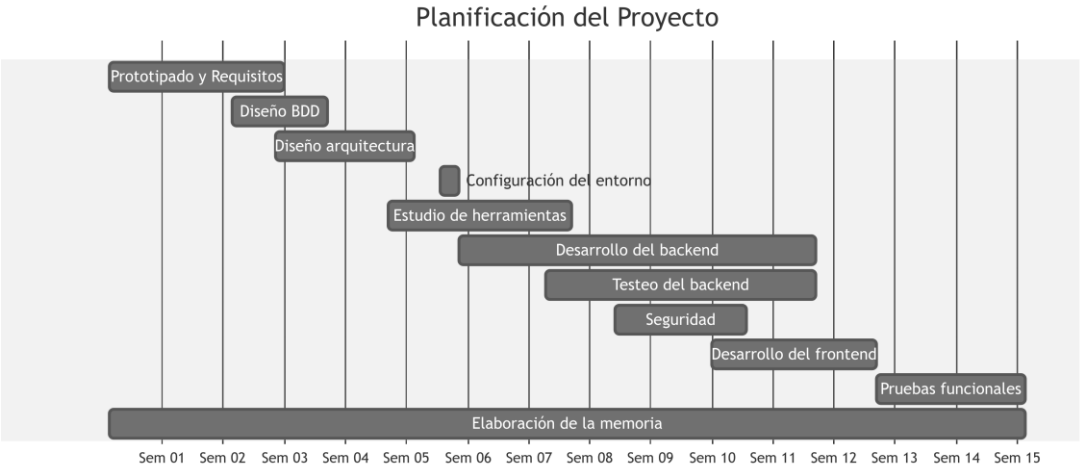

Figura 1: Diagrama de Gantt

La planificación del proyecto incluye una serie de tareas organizadas para maximizar la eficiencia mediante la ejecución simultánea de varias fases. Estas tareas, representadas en el diagrama de Gantt (véase *Figura 1*), son las siguientes:

1. Prototipado e identificación de requisitos.
2. Diseño de la base de datos (BDD).
3. Elaboración del diagrama de clases.
4. Diseño de la arquitectura del sistema.
5. Configuración del entorno de desarrollo.
6. Desarrollo del backend.
7. Testeo del backend.
8. Implementación de medidas de seguridad.
9. Desarrollo del frontend.
10. Realización de pruebas funcionales.
11. Elaboración de la memoria del proyecto.

La planificación del proyecto se centra en la ejecución eficiente de las tareas, combinando diferentes fases para aprovechar al máximo el tiempo disponible. El prototipado y la identificación de requisitos se llevarán a cabo junto con el diseño de la base de datos, facilitando la integración entre ambos elementos desde el inicio. A continuación, se elaborará un diagrama de clases, se diseñará la arquitectura del sistema y se configurará el entorno de desarrollo.

Conforme se adquieran las habilidades necesarias para manejar las herramientas, se procederá al desarrollo del backend, lo que permitirá comenzar el testeo de manera temprana y así detectar y corregir errores con agilidad. En paralelo, se trabajará en el desarrollo del frontend para garantizar una integración temprana de los diferentes componentes del sistema. La elaboración de la memoria será un proceso continuo, avanzando en los apartados a medida que se completen las fases técnicas, mientras que las pruebas funcionales se realizarán al final, asegurando que la plataforma esté completamente operativa.

1.4 Organización de la memoria

El resto de esta memoria se organiza en cinco capítulos. En el segundo capítulo, se aborda la selección de las tecnologías utilizadas para el desarrollo de la plataforma.

El tercer capítulo detalla el diseño de la propuesta, abarcando los requisitos funcionales y el diseño de la base de datos. El cuarto capítulo describe el desarrollo de la plataforma, incluyendo la implementación del backend, el desarrollo del frontend, la integración y el testeo de la aplicación.

En el quinto capítulo, se analizan los resultados obtenidos y se presentan las conclusiones, reflexionando sobre los logros alcanzados. Finalmente, el sexto capítulo evalúa el impacto del proyecto desde perspectivas personal, empresarial, social, económica, medioambiental y cultural, vinculándolo con los Objetivos de Desarrollo Sostenible (ODS) y proponiendo líneas de desarrollo futuro.

2 Tecnologías Empleadas

El proceso de selección de tecnologías para el desarrollo de esta aplicación web fue un paso crucial para garantizar un resultado eficiente, escalable y mantenible. A continuación, se detallan las opciones consideradas, las ventajas y desventajas de cada una, y las razones que llevaron a elegir las herramientas finales.

Frontend

Opciones consideradas

- **React**
  React [1]: React es una biblioteca ampliamente utilizada para construir interfaces de usuario gracias a su enfoque declarativo, basado en componentes. Una de sus principales ventajas es la flexibilidad que ofrece, ya que permite integrarse fácilmente con otras herramientas y librerías para personalizar la solución. Sin embargo, React no es un framework completo, lo que implica que es necesario añadir manualmente funcionalidades como el enrutado o el manejo de estados globales.

- **Next.js**
  Next.js [2]: Next.js es una herramienta basada en React que extiende sus capacidades al incorporar renderizado del lado del servidor (SSR), generación estática de páginas (SSG) y optimización automática de rutas. Estas características lo convierten en una solución ideal para aplicaciones web que requieren un excelente rendimiento y una buena experiencia de usuario. Sin embargo, su curva de aprendizaje es un poco más elevada debido a las múltiples opciones de renderizado que ofrece.

- **Angular**
  Angular [3]: Angular es un framework completo que incluye una solución integrada para muchas necesidades del desarrollo frontend, como enrutado, inyección de dependencias y pruebas. Aunque su enfoque "todo en uno" es beneficioso para proyectos grandes, puede ser excesivo para aplicaciones más simples. Además, su curva de aprendizaje es más pronunciada que la de React o Next.js.

Decisión final

Para este proyecto, se eligió Next.js debido a su capacidad para manejar múltiples estrategias de renderizado y su integración nativa con React. Esto permite construir una interfaz rápida, optimizada y con una experiencia de usuario mejorada. El uso de TypeScript se justificó por la necesidad de mejorar la mantenibilidad del código y prevenir errores comunes a través del tipado estático.

Backend

Opciones consideradas

- **Java Spring Boot**
  Spring Boot [4]: Spring Boot es un framework maduro y ampliamente adoptado en el desarrollo backend, reconocido por su robustez, flexibilidad y capacidad para construir servicios RESTful de manera eficiente. Su diseño modular facilita la implementación de aplicaciones escalables y de alto rendimiento, siendo especialmente adecuado para proyectos que requieren un control detallado sobre la arquitectura. Además, cuenta con un ecosistema extenso y bien documentado, que incluye herramientas para gestionar dependencias, seguridad, integración con bases de datos y despliegue. Estas características lo convierten en una opción ideal para desarrollar aplicaciones sostenibles y preparadas para evolucionar en entornos de mayor complejidad.

- **Django**
  Django [5]: Django, un framework desarrollado en Python, destaca por ofrecer una solución completa para el desarrollo backend, integrando de forma nativa herramientas como un ORM avanzado, gestión de rutas, autenticación y soporte para la creación de API REST mediante Django REST Framework. Estas características lo convierten en una opción eficiente para proyectos de pequeña o mediana escala, donde la rapidez en el desarrollo es un factor clave. Sin embargo, no fue seleccionado para este proyecto, ya que se optó por una solución que ofreciera un mayor control sobre la arquitectura y estuviera mejor preparada para garantizar la escalabilidad y el soporte de posibles escenarios futuros de mayor complejidad.

- **Node.js**
  Node.js [6]: Node.js permite construir aplicaciones backend con JavaScript, lo que facilita el uso de un solo lenguaje en toda la aplicación. Sin embargo, para este proyecto, las características y la escalabilidad proporcionadas por Spring Boot se consideraron más adecuadas.

Decisión final

Se optó por Java Spring Boot debido a su robustez y flexibilidad en el desarrollo de servicios backend, así como su capacidad para gestionar aplicaciones empresariales complejas. El ecosistema de Spring Boot, junto con su soporte para bibliotecas como JPA [7] (Java Persistence API), permite gestionar la capa de persistencia de manera eficiente sin necesidad de escribir consultas SQL manuales, como sería necesario con JDBC.

Base de datos

Opciones consideradas

- **MySQL**
  MySQL [8]: MySQL es una base de datos relacional ampliamente utilizada que ofrece un excelente equilibrio entre rendimiento, escalabilidad y facilidad de uso. Su compatibilidad con JPA simplifica la integración en aplicaciones basadas en Java Spring Boot.

- **MongoDB**
  MongoDB [9]: MongoDB, una base de datos NoSQL, se especializa en manejar datos no estructurados y permite gran flexibilidad en el diseño del esquema. Sin embargo, para este proyecto, que tiene una estructura de datos bien definida, una base de datos relacional como MySQL era más adecuada.

- **PostgreSQL**
  PostgreSQL [10]: PostgresSQL es una base de datos relacional avanzada que ofrece características adicionales como manejo de datos JSON. Aunque poderosa, su uso no se consideró necesario debido a que MySQL cubría adecuadamente las necesidades del proyecto.

Decisión final

Elegí MySQL por su simplicidad, rendimiento y excelente compatibilidad con JPA en Spring Boot. Además, su madurez y soporte en la comunidad fueron factores clave en la decisión.

Resumen

La elección de tecnologías en este proyecto se basó en criterios de escalabilidad, mantenibilidad y adecuación a las necesidades específicas de la aplicación. Mientras que Next.js y TypeScript garantizaron una experiencia de usuario óptima y un código frontend robusto, Java Spring Boot y MySQL proporcionaron una base sólida para el backend, asegurando un desarrollo eficiente y escalable. Todas estas decisiones se reflejan gráficamente en la *Figura 2.*

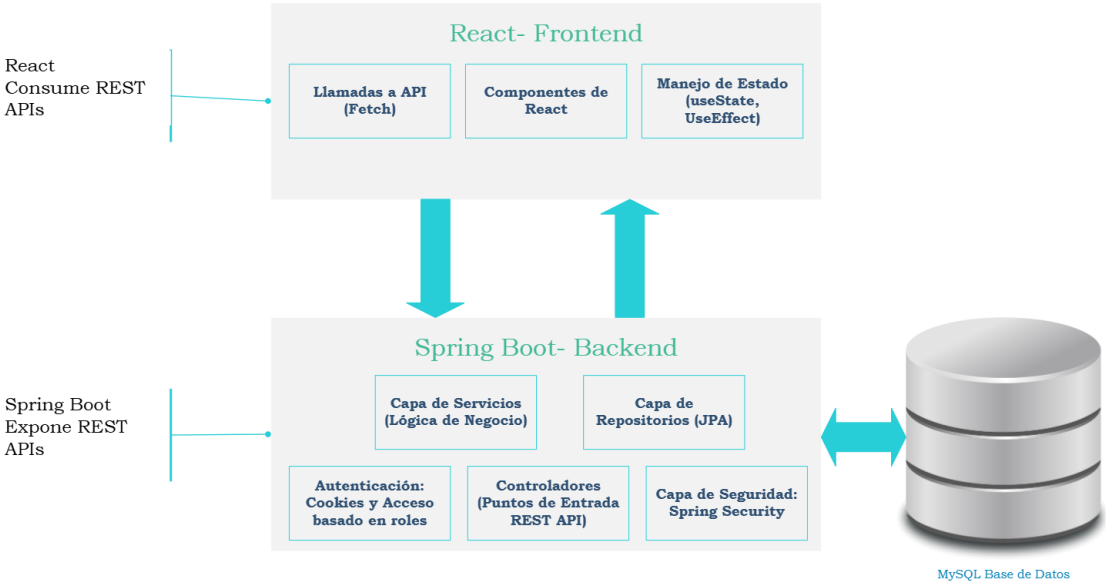

Figura 2: Arquitectura con Tecnologías

3 Diseño

A la hora de abordar el proyecto, se han tenido en cuenta las buenas prácticas y se han seguido estándares como el IEEE Std 830-1998 [11]. Es por esto por lo que en este apartado incluimos una sección de Diseño, con diagramas que faciliten la comprensión de las arquitecturas implementadas y con las debidas justificaciones de las decisiones tomadas.

3.1 Diseño de Alto Nivel

El Diseño de Alto Nivel es aquel que nos permite hacernos una vaga idea de las dimensiones del producto de software que tenemos la intención de desarrollar. Este paso es clave para entender las cualidades que debe satisfacer el producto e identificar qué funcionalidades son las más imprescindibles.

3.1.1 Especificación de Requisitos

Esta subsección es una Especificación de Requisitos Software (ERS). Tiene como propósito definir los requisitos y especificaciones del software a desarrollar. En otras palabras, tendremos una primera puesta en contacto con los objetivos que debe satisfacer nuestro producto.

3.1.1.1 Requisitos Funcionales

Los requisitos funcionales definen las capacidades y comportamientos del sistema, estableciendo las bases para su desarrollo. Estos se presentan a continuación en las *Tablas 1-14*, organizados en formato de tabla para facilitar su comprensión y consulta.

Tabla 1: Especificación de RF01

Identificación del Requisito	**RF01**
Nombre del requisito	Registro de aulas
Características	Permite introducir nuevas aulas y sus datos correspondientes
Actor	Administrador
Descripción	Las características de un aula deberán ser: • Identificador único • Nombre • Capacidad (número de personas) • Tamaño en m² • Foto • Descripción del aula

Tabla 2: Especificación de RF02

Identificación del Requisito	RF02
Nombre del requisito	Gestión de aulas
Características	Permite eliminar aulas
Actor	Administrador
Descripción	Si un aula deja de ser reservable, se podrá eliminar del sistema

Tabla 3: Especificación de RF03

Identificación del Requisito	RF03
Nombre del requisito	Consulta de aulas
Características	Permite la consulta de las aulas con sus respectivos datos
Actor	Usuarios
Descripción	El sistema mostrará todos los datos del aula en cuestión

Tabla 4: Especificación de RF04

Identificación del Requisito	RF04
Nombre del requisito	Reserva de aulas
Características	Permite reservar un espacio en una franja horaria determinada y a un único investigador
Actor	Usuarios
Descripción	Se debe seleccionar el aula en cuestión y especificar la fecha y hora en la que se desea hacer la reserva

Tabla 5: Especificación de RF05

Identificación del Requisito	RF05
Nombre del requisito	Consulta de reservas
Características	Los usuarios podrán acceder a los datos de sus reservas
Actor	Usuarios
Descripción	Se debe seleccionar la reserva que se quiere consultar y el sistema devolverá la información asociada a esta (fecha y hora, aula...)

Tabla 6: Especificación de RF06

Identificación del Requisito	**RF06**
Nombre del requisito	Gestión de reservas
Características	Permite eliminar reservas
Actor	Usuarios
Descripción	En caso de que un usuario no pueda asistir a su reserva, el sistema le permite cancelar la reserva para que otros puedan usarla

Tabla 7: Especificación de RF07

Identificación del Requisito	**RF07**
Nombre del requisito	Registro de usuarios
Características	Permite dar de alta nuevos investigadores con sus datos correspondientes
Actor	Usuarios
Descripción	Las características de un investigador deberán ser: • Correo electrónico (@ctb.upm.es) • Nombre • Apellido • Contraseña • Días de vacaciones • Días de asuntos propios • Rol (Investigador o Responsable), Si es Responsable: • Lista de usuarios que gestiona Si no es Responsable: • Lista de responsables que lo gestionan

Tabla 8: Especificación de RF08

Identificación del Requisito	**RF08**
Nombre del requisito	Gestión de usuarios
Características	El sistema debe permitir la gestión completa de las cuentas de usuario. Esto incluye la modificación de datos personales, la baja de usuarios y la gestión de contraseñas de manera segura.
Actor	Usuarios

Descripción	Los usuarios podrán gestionar sus propias cuentas, lo que incluye: • Modificar sus datos personales. • Actualizar su contraseña El administrador tendrá permisos adicionales, lo que le permitirá: • Modificar las relaciones responsable-subordinado de los usuarios • Adjudicar cuántos días de vacaciones y días de asuntos propios le corresponden a cada usuario • Dar de baja

Tabla 9: Especificación de RF09

Identificación del Requisito	**RF09**
Nombre del requisito	Solicitud de ausencias
Características	Permite solicitar ausencias
Actor	Usuarios
Descripción	Los usuarios podrán seleccionar los días en los que desean ausentarse (vacaciones, asuntos propios o baja por enfermedad). En casos excepcionales, podrán solicitar más días de los que les corresponden y el sistema emitirá una advertencia tanto al usuario como al responsable que autoriza la ausencia. Para su correcto funcionamiento, además, el sistema debe • Excluir automáticamente los fines de semana y días festivos (ver RF13) del cálculo de las ausencias solicitadas. • Disponer de contadores que se asegurarán, actualizándose en tiempo real tras cada solicitud aprobada, de que los días solicitados no excedan los que corresponden a cada usuario. Una ausencia consta de: • Día de comienzo • Día de fin • Tipo de ausencia (Días de asuntos propios / Días de vacaciones / Baja por enfermedad)) • Estado de la solicitud (Aceptada/Rechazada/Pendiente)

Tabla 10: Especificación de RF10

Identificación del Requisito	RF10
Nombre del requisito	Gestión de ausencias
Características	Permite cancelar y modificar las solicitudes de ausencia
Actor	Usuarios
Descripción	Se pueden hacer sobre las ausencias que estén en estado “Pendiente”

Tabla 11: Especificación de RF11

Identificación del Requisito	RF11
Nombre del requisito	Consulta de ausencias
Características	Permite que consulten las ausencias de los usuarios que tienen bajo su dirección para facilitar la coordinación de los equipos
Actor	Usuarios con rol de responsable
Descripción	El sistema permitirá a los responsables consultar las fechas en las que los miembros de su equipo están ausentes, ya sea por vacaciones, asuntos propios o baja por enfermedad. Los usuarios regulares solo podrán consultar sus propias ausencias

Tabla 12: Especificación de RF12

Identificación del Requisito	RF12
Nombre del requisito	Gestión de solicitudes de ausencias
Características	Permite aprobar o rechazar las solicitudes de ausencia de los usuarios que tienen bajo su dirección
Actor	Usuarios con rol de responsable
Descripción	El sistema deberá mostrar las ausencias pendientes de aprobación, incluyendo información sobre el tipo de ausencia y las fechas solicitadas. Los responsables podrán: • Aprobar la solicitud. • Denegar la solicitud: los responsables podrán denegar una solicitud si lo consideran necesario. La solicitud quedará marcada como "Rechazada" en el sistema y no afectará a los

	contadores de días de vacaciones o asuntos propios. En cualquier caso, se notificará al solicitante

Tabla 13: Especificación de RF13

Identificación del Requisito	RF13
Nombre del requisito	Registro de días festivos
Características	Permite registrar los días festivos
Actor	Administrador
Descripción	El administrador podrá registrar los días festivos anuales. Estos días no deben ser contados en el cálculo de las ausencias (vacaciones, asuntos propios o baja por enfermedad) de los usuarios

3.1.1.2 Interfaces Externas

Tabla 14: Especificación de RI01

Identificación del Requisito	RI01
Nombre del requisito	Interfaz Gráfica
Características	La interfaz gráfica deberá mostrar calendarios actualizables con las reservas y/o bajas de los usuarios.
Descripción	Desde la interfaz gráfica se deberá permitir realizar cualquier tipo de operaciones. Por ejemplo, al seleccionar un aula en la lista de aulas, el usuario deberá poder ver detalles adicionales, como las franjas horarias disponibles para su reserva

Para la visualización de la Interfaz Gráfica se ha optado por elaborar un prototipo de alta fidelidad (*HiFi*) de la página web, del cual se pueden ver capturas en el apartado Anexo. Este prototipo se ha elaborado a través de la plataforma *Figma*.

3.1.2 Diagrama de Contexto Final

El diagrama de contexto final nos ayuda a tener una visión global del sistema, es decir, nos permite visualizar las interacciones entre sus componentes clave. Este diagrama destaca las entradas y salidas principales, permitiéndonos ver

cómo fluyen los datos y cómo se comportan los sistemas implicados para satisfacer los requisitos definidos en el apartado anterior.

Para su elaboración, vamos a comenzar por identificar los actores externos, que en nuestro caso son:

- **Usuario del Sistema (Investigador)**: personas que usan el sistema para registrar ausencias (vacaciones, asuntos propios o baja por enfermedad), reservar aulas y consultar información. Los usuarios pueden tener el rol de Subordinado o Responsable. Los responsables tienen la capacidad de aprobar o denegar las solicitudes de vacaciones de los investigadores que gestionan.

- **Administrador**: usuario con privilegios que puede gestionar aulas (crear, modificar y eliminar), así como cuentas de usuarios y establecer cuáles son los días festivos en el sistema.

- **Sistema de Autenticación**: se encarga del proceso de inicio de sesión y autenticación de usuarios.

- **Sistema de Correo**: se encarga de notificar por correo a los usuarios cuando hacen una reserva o se produce un cambio en el estado de la petición de sus ausencias. También avisa a los usuarios cuando tienen una reserva con quince minutos de antelación.

El segundo paso sería identificar los casos de uso en los que participan las entidades identificadas:

- **Reservar Aula**: el usuario reserva un aula para una fecha y hora específicas.

- **Cancelar Reserva**: el usuario cancela una reserva previamente hecha.

- **Solicitar Ausencias**: los investigadores solicitan días de vacaciones, asuntos propios o bajas por enfermedad.

- **Modificar Ausencias**: el usuario puede borrar una petición de ausencia.

- **Consultar Ausencias**: los usuarios con rol de responsable pueden consultar las ausencias (vacaciones, asuntos propios o baja por

enfermedad) de los miembros de su equipo. Los usuarios regulares pueden consultar sus propias peticiones de ausencia.

- **Recibir Notificación de Confirmación**: el sistema envía por correo una notificación al usuario confirmando acciones como reservas realizadas o solicitudes de ausencias procesadas.

- **Recibir Notificaciones de Recordatorio**: el sistema avisa a los usuarios cuando se aproxima su reserva para que no se olviden de esta.

- **Gestionar Aulas**: el administrador puede dar de alta y eliminar aulas o espacios disponibles para reserva.

- **Gestionar Usuarios**: el administrador puede dar de baja a usuarios o modificar sus datos (rol, días de asuntos personales etc.).

- **Registrar Días Festivos**: el administrador puede definir los días festivos del año, que no contarán como días de ausencia ni para las reservas de aulas.

- **Registrarse**: los usuarios pueden crear una cuenta usando un formulario de registro.

- **Modificar Datos Personales**: los usuarios pueden modificar sus datos personales, como el nombre, correo electrónico y contraseña.

- **Autenticación** (Sistema de Autenticación): el sistema valida el acceso de los usuarios.

A continuación, vamos a identificar los almacenes de datos necesarios para satisfacer todos los procesos mencionados:

- **Aulas**: es la fuente de datos para la gestión y consulta de espacios disponibles en el sistema.

- **Reservas**: almacena datos relacionados con las reservas de los espacios, asegurando que no se sobreponen en una misma franja horaria.

- **Usuarios**: es la fuente de datos encargada de mantener los registros de los usuarios que utilizan el sistema.

- **Ausencias**: almacena los datos relacionados con las solicitudes de ausencias de los usuarios, incluyendo el tipo de ausencia, las fechas, el estado y la persona que debe autorizarlas.

- **Festivos**: es una colección de los días que constan como festivos

Finalmente, la *Figura 3* muestra el diagrama de contexto que captura todo lo anterior.

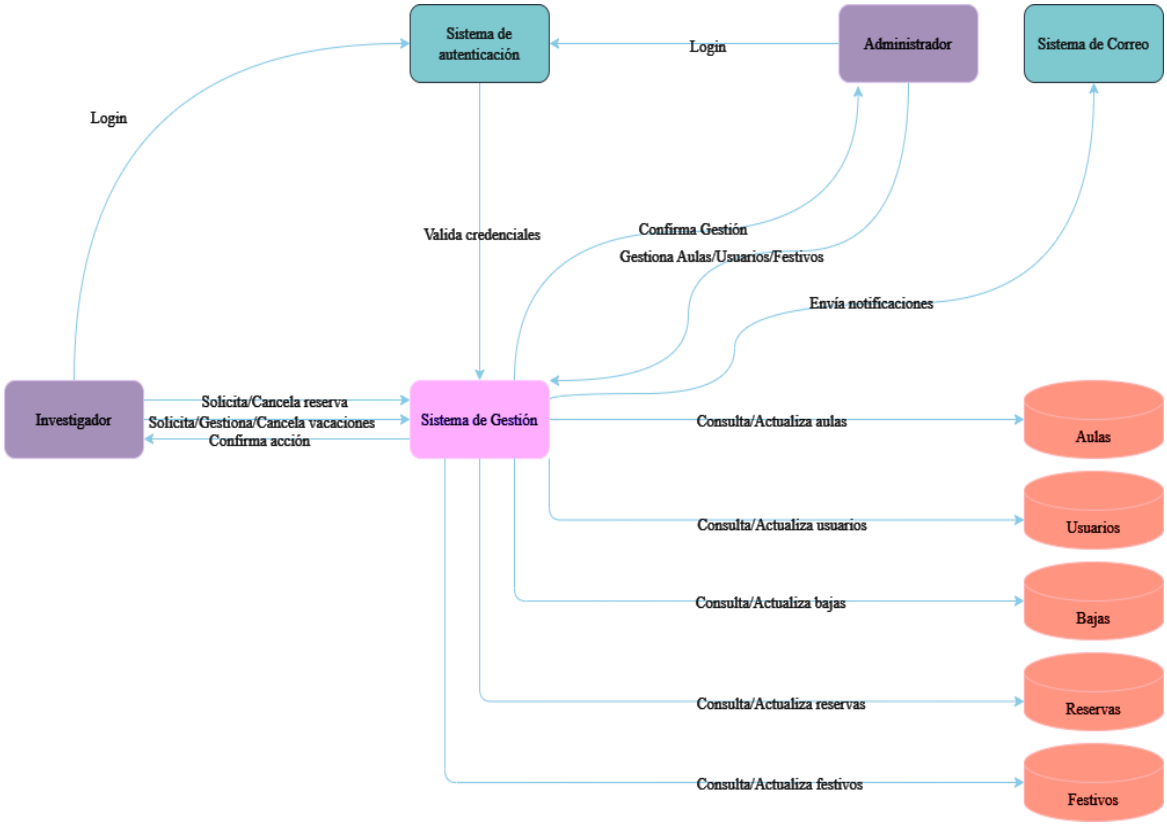

Figura 3: Diagrama de Contexto Final

3.2 Diseño de Bases de Datos

Las bases de datos son la forma de almacenar y gestionar información dentro de una empresa o ámbito. Para el proyecto se ha optado por hacer un diseño relacional, lo que significa que los datos se organizan en tablas (o relaciones), donde cada fila representa un registro y cada columna un campo. Bases de Datos Relacionales [12]: Este modelo fue conceptualizado por primera vez en los años 70 por Edgar F. Codd y resultó revolucionario por su capacidad de realizar consultas complejas utilizando el lenguaje *SQL* (*Structured Query Language*).

Esta opción es adecuada para el proyecto, sobre todo teniendo en cuenta que la naturaleza de los datos que vamos a manejar es estructurada y requiere de consistencia. Para más información, véase el *Capítulo 2: Tecnologías Empleadas.*

3.2.1 Diagrama Entidad-Relación

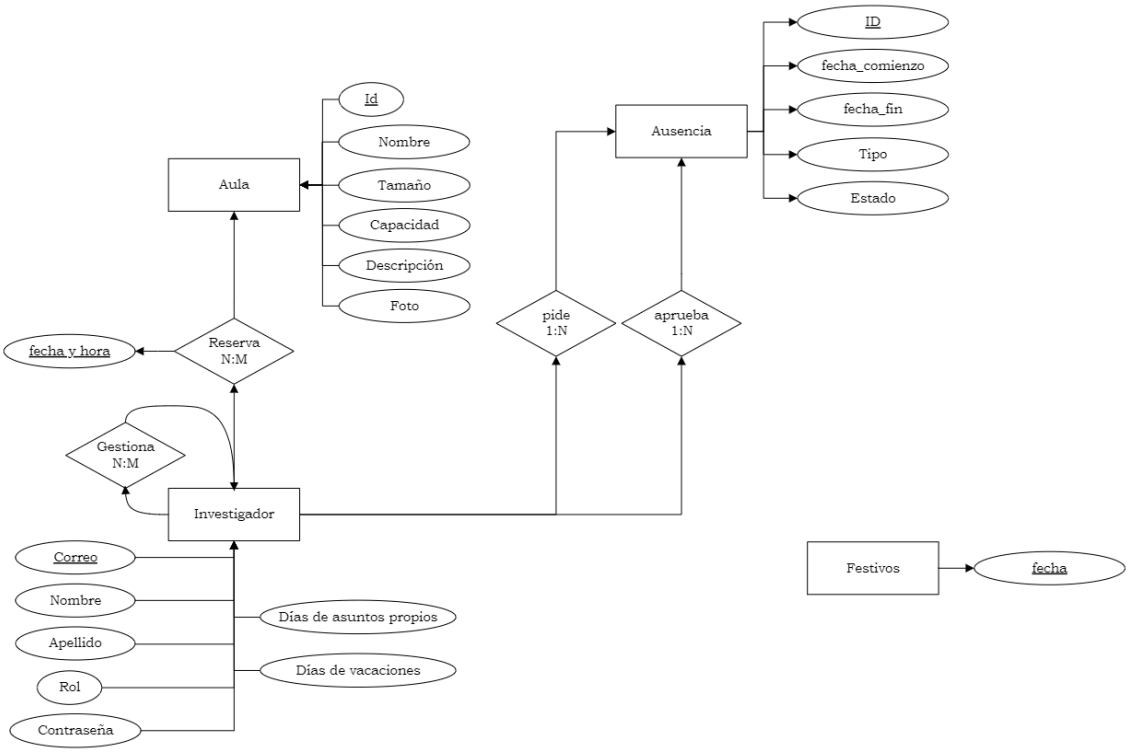

Figura 4: Diagrama Entidad-Relación

Tal y como se puede observar en la *Figura 4*, se han identificado las siguientes entidades y relaciones clave en el Modelo Entidad-Relación (ER), que es una forma de diseñar bases de datos relacionales de manera gráfica:

En primer lugar, tenemos la entidad **Aula**, con atributos *id* (esta sería la llave primaria, es decir, la forma de identificar aulas en el sistema), *Nombre* (por ejemplo "Aula 1025" o "Salón de Actos 1"), *Tamaño* (una estimación en metros cuadrados del espacio), *Capacidad* (una estimación de cuántas personas pueden reunirse y trabajar en dicho espacio), además de una breve *descripción* (en la que se podría detallar el material que hay disponible, por ejemplo proyectores o cantidad de enchufes) y una *foto* que ayude a los usuarios a identificar el aula que quieren reservar visualmente.

Esta entidad está conectada con la entidad **Investigador** en una relación de muchos a muchos (N:M) lo cual significa que cualquier investigador puede reservar cualquier aula en una franja de tiempo determinada (atributo *fecha y hora* de la relación *Reserva*), con la condición de que esta no esté reservada por otro usuario (esto lo garantizaremos a través del *front-end*, es decir, la propia interfaz gráfica no permitirá a los usuarios hacer una petición de reserva a un aula ya reservada).

Por su parte, la entidad **Investigador** tiene atributos *Correo* (que debe pertenecer al dominio de la organización y que sirve para identificarle en el sistema), *Nombre, Apellido, Rol* (que le asigna el *Administrador del sistema* y que puede ser *Responsable* o *Subordinado*. La distinción principal entre estos dos roles es que los *Responsables* coordinan sus equipos y pueden aprobar las ausencias de sus subordinados), además tiene los atributos *Días de asuntos propios* y *Días de vacaciones* (que les asigna el administrador y cuyo valor predeterminado es 7 y 15 y que se van descontando según su responsable les

aprueba las solicitudes. En caso de alcanzar el límite, es decir que queden cero de cada tipo, el sistema permite al usuario hacer la petición de ausencia, pero avisa a través de una advertencia tanto al subordinado como a su respectivo responsable). Finalmente está el atributo *contraseña*, que en la base de datos se guardará como un hash, o sea se cifrará, por motivos de seguridad y que se usará para el inicio de sesión.

Investigador tiene dos relaciones de uno a muchos (1:N) con la entidad **Ausencia**, la primera *pide* la hace con el rol de subordinado (un subordinado puede solicitar varias bajas) y, la segunda *aprueba* la hace con el rol de responsable (un responsable puede aprobar o rechazar varias propuestas de ausencia de sus subordinados. Además **Investigador** tiene una relación de muchos a muchos (N:M) consigo ya que un responsable puede coordinar a varias personas y un subordinado puede tener varios coordinadores.

Una **Ausencia**, por su parte, tiene los atributos *ID, fecha de inicio, fecha fin*, *Tipo* (este va a ser un enumerado de opciones, tales como "Baja por Enfermedad", "Días de asuntos personales" con la posibilidad de añadirse más a futuro). También dispone del atributo *Estado* que permite clasificarlas en *Aceptada* (aquellas aprobadas por el responsable), *Rechazada* (aquellas que fueron denegadas) y *Pendiente* (aquellas que aún no han tenido respuesta por parte del responsable).

Finalmente, tenemos la entidad **Festivos** que dispone de un único atributo *fecha,* y de ninguna relación, ya que se usará solo por el Administrador para gestionar el conteo de las ausencias del personal debidamente.

3.2.2 Modelo Lógico de Datos

Un poco más a bajo nivel, podemos hacer un Modelo Lógico de Datos (*Figura 5)* para guiarnos en el desarrollo de la Base de Datos. Este diagrama es una versión más detallada del que podemos ver en la *Figura 4*, pues contiene información sobre el tipado de los datos que se guardan (es decir, si se trata de un entero o una cadena alfanumérica entre otros) y también revela las claves foráneas (es decir, las que van a servir de puentes para relacionar a dos entidades entre sí). Para hacer dicho modelo, se ha empleado la herramienta My*SQL Workbench*.

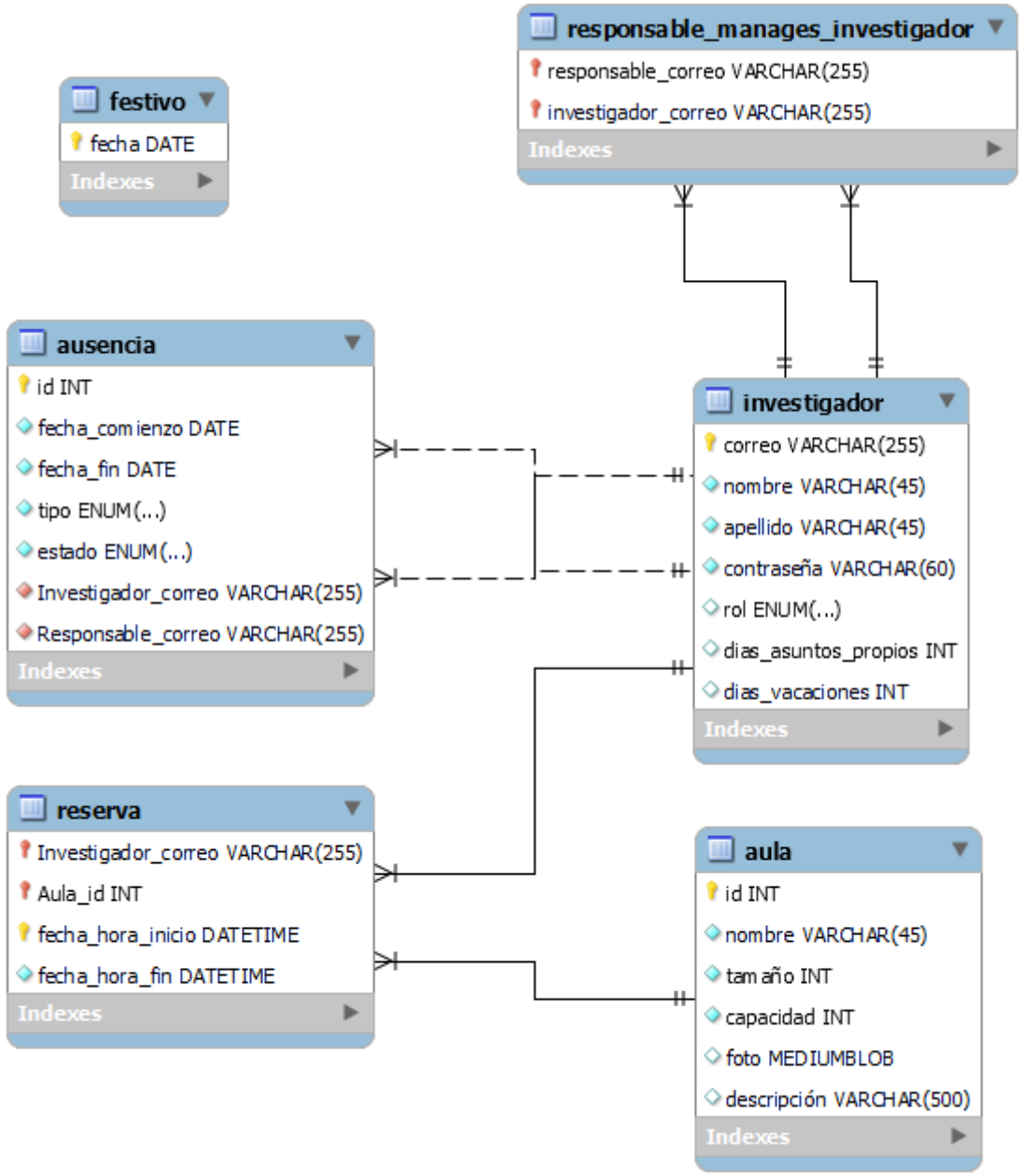

Figura 5: Modelo Lógico de Datos

3.3 Diseño de Bajo Nivel

El diseño de Bajo Nivel es una aproximación más cercana a lo que va a ser la producción de código para el proyecto. Sirve para establecer la forma del repositorio, las clases que va a contener y cómo se relacionarán entre sí.

A continuación, en la *Figura 5* y *Figura 6* se puede observar lo que se denomina comúnmente como un **diagrama de clases**, una herramienta cuyo objetivo es ilustrar las relaciones entre las diferentes clases que serán programadas en el proyecto. El diseño se ha llevado a cabo utilizando la herramienta *Mermaid*.

Como se puede apreciar, tenemos la clase *API* que servirá como puente entre el front-end (la interfaz gráfica) y el back-end (las funcionalidades) del proyecto. A

través de esta capa podremos hacer llamadas a los distintos servicios (*FestivoService, AulaService, UsuarioService, ReservaService, AusenciaService, InvestigadorAuthService, NotificationService, ReservaReminderService*), algunos de estos hacen operaciones CRUD [13] (por las siglas en inglés *Create Read Update* Delete), es decir, estas clases pueden añadir, leer, modificar o borrar objetos de la Base de Datos.

Los servicios *NotificationService* y *ReservaReminderService* consistirán en llamar a otra API de correos que se encargue de mandar las notificaciones y advertencias acordadas en el apartado Especificación de Requisitos. Este último, además, hará comprobaciones periódicas de cuáles son las reservas que comienzan en los próximos quince minutos y notificará a los usuarios involucrados.

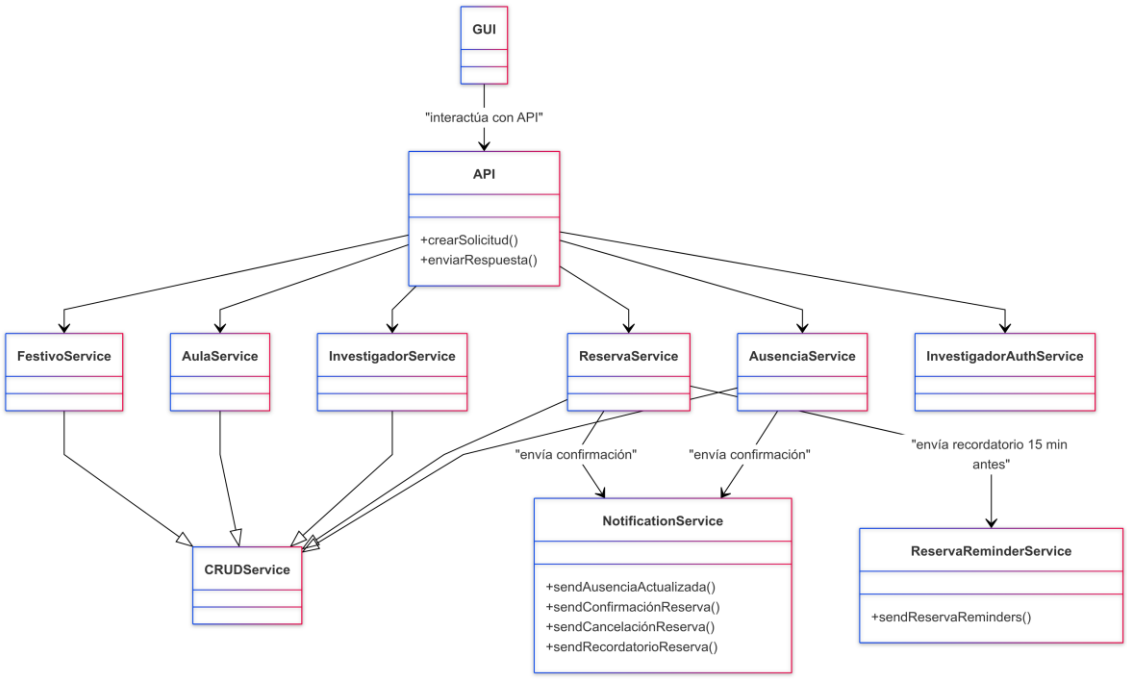

Figura 6: Diagrama de Clases I

El servicio de *InvestigadorAuthService* se ayuda de una clase *PasswordEncoder* cuyo fin es cifrar y validar las contraseñas guardadas en el sistema. El objetivo es autenticar a los usuarios del sistema (clase *Investigador).*

Los investigadores tienen un rol asociado, y según este pueden hacer diferentes peticiones al sistema:

> El *Administrador* puede hacer llamadas a prácticamente todos los servicios del sistema, aunque no están explícitamente dibujadas las relaciones (por mantener la claridad y legibilidad en el diagrama).
>
> Tanto el *Responsable* como el *Subordinado* pueden hacer peticiones a las clases *Reserva* y *Ausencia,* para reservar aulas o pedir vacaciones. El *Responsable,* además, puede aprobar o denegar las ausencias de sus subordinados (no dibujado en el diagrama por simplicidad).

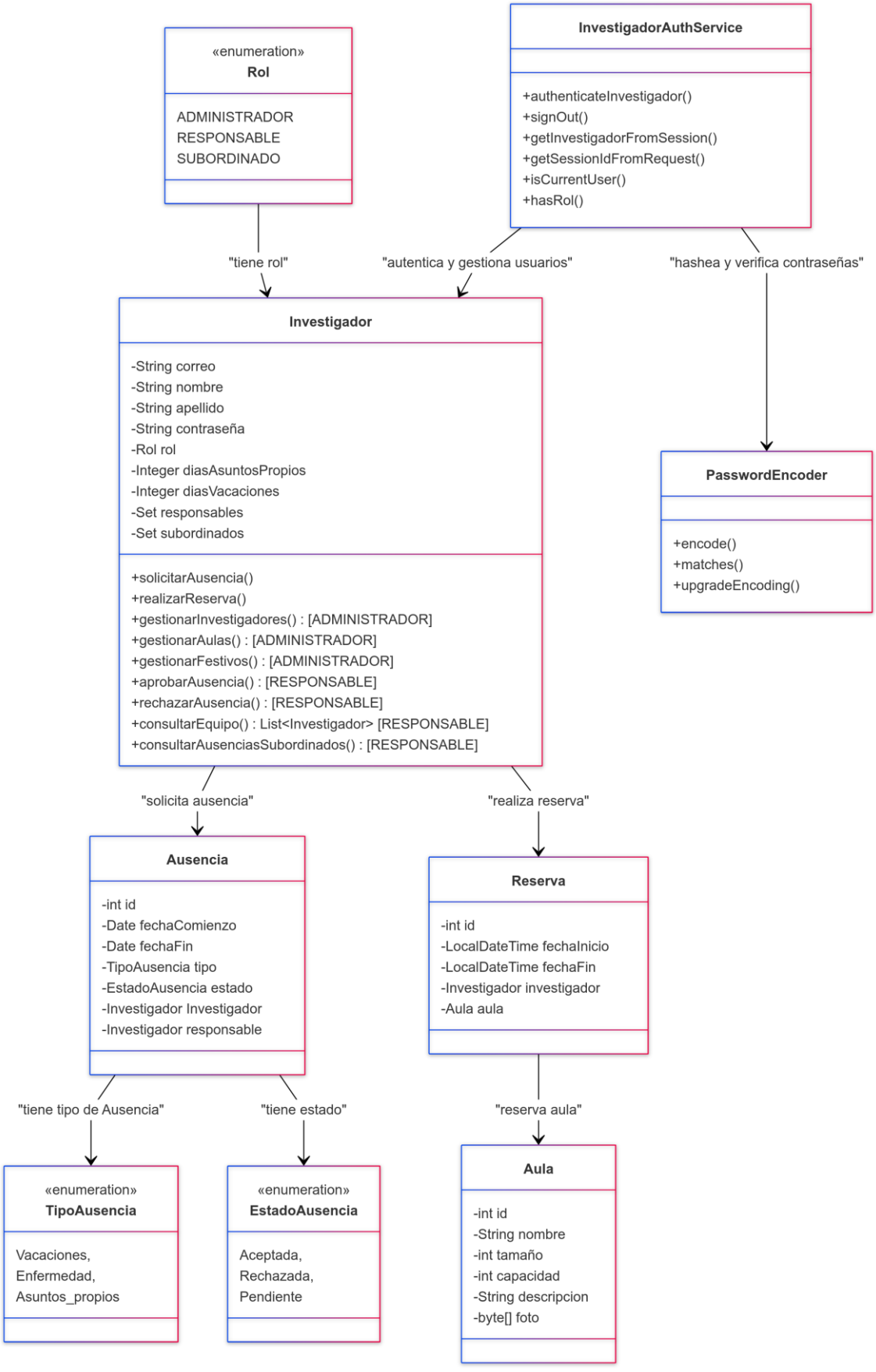

Figura 7: Diagrama de Clases II

Para terminar de definir el sistema, se ha optado por separar por capas el servicio (véase la *Figura 8*), e implementar una arquitectura de tres capas, siguiendo las prácticas recomendadas por IBM.

Three-Tier Architecture [14]: La primera capa o *Presentation Tier* representa la interfaz y la capa de comunicación con el usuario, y su funcionalidad principal es mostrar y recolectar información de este. La petición en forma de objeto JSON proporciona al sistema un *DTO* (*Data Transfer Object*) con los campos necesarios para interactuar con el sistema, este DTO se traduce a un objeto con los campos de la entidad que representa a través de las clases Mapper. Esta capa consistirá sobre todo de *Controllers* o puntos de acceso *REST* que se encargarán de recibir y procesar peticiones *HTTP*.

Después, está la capa *Application Tier* que es el corazón de la aplicación. En esta la información que se recolectó en la *Presentation Tier* se procesa usando la lógica del negocio, aunque también puede agregar, modificar o eliminar información en la *Data Tier.*

La última capa, *Data Tier*, como su nombre indica es donde la información procesada por la aplicación se guarda, como la Base de Datos u otras API.

Figura 8: Diagrama de Capas

Esta forma de hacer las cosas tiene muchos beneficios, en especial la separación lógica y física de la funcionalidad. Cada capa puede implementarse en sistemas y plataformas distintas que estén optimizadas para satisfacer sus requisitos funcionales. Y, al haber separación, cada capa puede modificarse sin afectar a las otras.

Así pues, cada servicio de la *Figura 6,* en el repositorio se verá subdividido en tres clases distintas, por ejemplo, habrá una clase *AulaController* recibiendo las peticiones HTTP que se hacen a las aulas, una clase *AulaService* con la lógica del servicio y una clase *AulaRepository* donde estarán las consultas o *queries* a la Base de Datos.

4 Desarrollo y Testeo

4.1 Desarrollo de Backend

En este apartado se proporciona una descripción general de los distintos endpoints de las API REST con ejemplos de JSON. La información está agrupada según el propósito de cada API, con ejemplos de JSON formateados para cada endpoint.

4.1.1 API Investigadores

1. **GET /investigadores/{correo}**
   - **Descripción**: obtiene la información de un investigador por su correo electrónico.
   - **Ejemplo de Petición**:

     ```
     GET
     http://localhost:8080/investigadores/newuser7@ctb.upm.es
     ```

   - **Ejemplo de Respuesta:**

     ```
     {
       "correo": "newuser7@ctb.upm.es",
       "nombre": "Jane",
       "apellido": "Smith",
       "rol": "Subordinado",
       "diasAsuntosPropios": 4,
       "diasVacaciones": 8,
       "responsables": ["newuser2@ctb.upm.es",
     "newuser3@ctb.upm.es"]}
     ```

2. **POST /investigadores**
   - **Descripción:** crea un nuevo investigador.
   - **Ejemplo de Petición:**

     ```
     POST http://localhost:8080/investigadores
     ```

   - **Cuerpo de la Petición:**

     ```
     {
       "correo": "newuser8@ctb.upm.es",
       "nombre": "John",
       "apellido": "Doe",
       "rol": "Subordinado",
       "diasAsuntosPropios": 5,
       "diasVacaciones": 10,
       "responsables": ["newuser2@ctb.upm.es",
     "newuser3@ctb.upm.es"]
           }
     ```

3. **GET /investigadores**
   - **Descripción:** obtiene una lista de todos los investigadores.
   - **Ejemplo de Petición:**

```
GET http://localhost:8080/investigadores
```

- **Ejemplo de Respuesta:**

```
[
  {
    "correo": "newuser1@ctb.upm.es",
    "nombre": "Alice",
    "apellido": "Smith",
    "contraseña": "$2a$10$...",
    "rol": "Responsable",
    "diasAsuntosPropios": 4,
    "diasVacaciones": 8,
    "responsables": ["newuser2@ctb.upm.es",
"newuser3@ctb.upm.es"]}
  {
    "correo": "newuser2@ctb.upm.es",
    "nombre": "Bob",
    "apellido": "Johnson",
    "contraseña": "$2a$10$...",
    "rol": " Responsable",
    "diasAsuntosPropios": 3,
    "diasVacaciones": 7,
    "responsables": ["newuser1@ctb.upm.es",
"newuser3@ctb.upm.es"]}
      ]
```

4. **GET /api/investigadores/responsable/{correoResponsable}**
   - **Descripción:** obtiene una lista de investigadores bajo un responsable específico.
   - **Ejemplo de Petición:**

```
GET
http://localhost:8080/investigadores/responsable/newuser1@
ctb.upm.es
```

   - **Ejemplo de Respuesta:**

```
[
  {
    "correo": "newuser2@ctb.upm.es",
    "nombre": "Bob",
    "apellido": "Johnson",
    "contraseña": "$2a$10$...",
    "rol": "Subordinado",
    "diasAsuntosPropios": 3,
    "diasVacaciones": 7,
    "responsables": ["newuser1@ctb.upm.es"]
  }
      ]
```

5. **PATCH /api/investigadores/{correo}**
   - **Descripción**: actualiza la información de un investigador.
   - **Ejemplo de Petición:**

```
PATCH
http://localhost:8080/investigadores/newuser7@ctb.upm.es
```

- **Cuerpo de la Petición:**

```
{
  "nombre": "Jane",
  "apellido": "Smith",
  "diasAsuntosPropios": 5
      }
```

- **Ejemplo de Respuesta:**

```
{
  "correo": "newuser7@ctb.upm.es",
  "nombre": "Jane",
  "apellido": "Smith",
  "contraseña": "$2a$10$...",
  "rol": "Subordinado",
  "diasAsuntosPropios": 5,
  "diasVacaciones": 8,
  "responsables": ["newuser2@ctb.upm.es"]
      }
```

6. **DELETE /api/investigadores/{correo}**
   - **Descripción**: elimina un investigador por su correo electrónico.
   - **Ejemplo de Petición:**

```
DELETE
http://localhost:8080/investigadores/newuser7@ctb.upm.es
```

   - **Ejemplo de Respuesta:**

```
HTTP/1.1 204 No Content
```

4.1.2 API Reservas

1. **POST /api/reservas**
   - **Descripción**: crea una nueva reserva.

   - **Cuerpo de la Petición:**

```
{
  "investigador": "newuser7@ctb.upm.es",
  "aula": 101,
  "fechaHoraInicio": "2024-10-17T10:00:00",
  "fechaHoraFin": "2024-10-17T12:00:00"
      }
```

2. **GET /api/reservas/aulas/{aulaId}**
   - **Descripción**: obtiene las reservas para un aula específica por su ID.

   - **Ejemplo de Respuesta:**

```
[
  {
    "investigador": "newuser7@ctb.upm.es",
    "aula": 101,
```

```
    "fechaHoraInicio": "2024-10-17T10:00:00",
    "fechaHoraFin": "2024-10-17T12:00:00"
  },
  {
    "investigador": "newuser8@ctb.upm.es",
    "aula": 101,
    "fechaHoraInicio": "2024-10-18T10:00:00",
    "fechaHoraFin": "2024-10-18T12:00:00"
  }
      ]
```

3. **GET /api/reservas/by-id**
   - **Descripción**: obtiene una reserva específica por su ID compuesto (correo del investigador, ID del aula, y fecha/hora de inicio).

   - **Ejemplo de Petición:**

     ```
     GET
     http://localhost:8080/reservas/by-
     id?investigadorCorreo=newuser7@ctb.upm.es&aulaId=101&fecha
     HoraInicio=2024-10-17T10:00:00
     ```

   - **Ejemplo de Respuesta:**

     ```
     {
       "investigador": "newuser7@ctb.upm.es",
       "aula": 101,
       "fechaHoraInicio": "2024-10-17T10:00:00",
       "fechaHoraFin": "2024-10-17T12:00:00"
           }
     ```

4. **GET /api/reservas/usuarios/{investigadorCorreo}**
   - **Descripción**: obtiene todas las reservas realizadas con el correo del investigador pasado como parámetro.

   - **Ejemplo de Petición:**

     ```
     GET
     http://localhost:8080/reservas/usuarios/user7@ctb.upm.es
     ```

   - **Ejemplo de Respuesta:**

     ```
     {
       "investigador": "user7@ctb.upm.es",
       "aula": 101,
       "fechaHoraInicio": "2024-10-17T10:00:00",
       "fechaHoraFin": "2024-10-17T12:00:00"
           }
     ```

5. **DELETE /api/reservas/by-id**
   - **Descripción**: elimina una reserva específica por su ID compuesto.

   - **Ejemplo de Petición:**

     ```
     DELETE
     ```

```
http://localhost:8080/reservas/by-
id?investigadorCorreo=newuser7@ctb.upm.es&aulaId=101&fecha
HoraInicio=2024-10-17T10:00:00
```

- **Ejemplo de Respuesta:**

```
HTTP/1.1 200 OK "Reserva eliminada correctamente"
```

6. **DELETE /api/reservas/before-date**
   - **Descripción**: elimina todas las reservas anteriores a una fecha específica.

   - **Ejemplo de Petición:**

```
DELETE
http://localhost:8080/reservas/before-date?fecha=2024-10-
01T00:00:00
```

   - **Ejemplo de Respuesta:**

```
HTTP/1.1 200 OK "Reservas eliminadas correctamente"
```

4.1.3 API Ausencias

1. **POST /api/ausencias**
   - **Descripción**: crea una nueva ausencia. Al principio se crea con el campo *responsable* a null y es cuando se acepta que se guarda el correo del usuario que la ha aceptado.

   - **Cuerpo de la Petición:**

```
{
  "fechaComienzo": "2024-10-01",
  "fechaFin": "2024-10-10",
  "tipo": "Vacaciones",
  "estado": "Pendiente",
  "investigador": "newuser7@ctb.upm.es",
      }
```

2. **GET /api/ausencias/{userCorreo}**
   - **Descripción**: obtiene todas las ausencias de un investigador específico por su correo electrónico.

   - **Ejemplo de Petición:**

```
GET http://localhost:8080/ausencias/newuser7@ctb.upm.es
```

   - **Ejemplo de Respuesta:**

```
[
  {
    "id": 1,
    "fechaComienzo": "2024-10-01",
    "fechaFin": "2024-10-10",
    "tipo": "Vacaciones",
```

```
    "estado": "Aceptada",
    "investigador": "newuser7@ctb.upm.es",
    "responsable": "newuser2@ctb.upm.es"
  }
      ]
```

3. **GET /api/ausencias/responsable/{responsableCorreo}**
   - **Descripción**: obtiene todas las ausencias de los subordinados de un responsable.

   - **Ejemplo de Petición:**

```
GET
http://localhost:8080/ausencias/responsable/newuser2@ctb.upm.es
```

   - **Ejemplo de Respuesta:**

```
[
  {
    "id": 1,
    "fechaComienzo": "2024-10-01",
    "fechaFin": "2024-10-10",
    "tipo": "Vacaciones",
    "estado": "Aceptada",
    "investigador": "newuser7@ctb.upm.es",
    "responsable": "newuser2@ctb.upm.es"
  }
      ]
```

4. **PATCH /api/ausencias/{id}**
   - **Descripción**: actualiza una ausencia por su ID. Hay dos tipos: cuando se invoca por el responsable de la ausencia para autorizar/denegarla y cuando se invoca por el investigador que pidió su ausencia para cambiar, por ejemplo, la fecha de inicio.

   - **Ejemplo de Petición:**

```
PATCH http://localhost:8080/ausencias/1
```

   - **Cuerpo de la Petición:**

```
{
  "estado": "Aceptada",
  "responsable": "newuser2@ctb.upm.es"

      }
```

   - **Ejemplo de Respuesta:**

```
{
  "id": 1,
  "fechaComienzo": "2024-10-01",
  "fechaFin": "2024-10-10",
  "tipo": "Vacaciones",
  "estado": "Aceptada",
  "investigador": "newuser7@ctb.upm.es",
```

```
  "responsable": "newuser2@ctb.upm.es"
      }
```

5. **DELETE /api/ausencias/{id}**
   - **Descripción**: elimina una ausencia por su ID.
   - **Ejemplo de Petición:**

     ```
     DELETE http://localhost:8080/ausencias/1
     ```

   - **Ejemplo de Respuesta:**

     ```
     HTTP/1.1 204 No Content
     ```

4.1.4 API Festivos

1. **POST /api/festivos**
   - **Descripción**: crea un nuevo día festivo.
   - **Cuerpo de la Petición:**

     ```
     {
       "fecha": "2024-12-25"
           }
     ```

2. **GET /api/festivos**
   - **Descripción**: obtiene una lista de todos los días festivos.
   - **Ejemplo de Petición:**

     ```
     GET http://localhost:8080/festivos
     ```

   - **Ejemplo de Respuesta:**

     ```
     [
       {
         "fecha": "2024-12-25"
       },
       {
         "fecha": "2024-01-01"
       }
           ]
     ```

3. **DELETE /api/festivos/by-fecha**
   - **Descripción**: elimina un día festivo por su fecha.
   - **Ejemplo de Petición:**

     ```
     DELETE http://localhost:8080/festivos/by-fecha?fecha=2024-
     12-25
     ```

   - **Ejemplo de Respuesta:**

     ```
     HTTP/1.1 200 OK "Festivo eliminado correctamente"
     ```

4.1.5 API Aulas

1. **POST /api/aulas**
   - **Descripción**: crea una nueva aula.

   - **Cuerpo de la Petición:**

```
{
  "nombre": "Aula 101",
  "tamaño": 50,
  "capacidad": 20,
  "foto": null,
  "descripción": "Aula equipada con proyector"
      }
```

2. **GET /api/aulas/{id}**
   - **Descripción**: obtiene los detalles de un aula específica por su ID.

   - **Ejemplo de Petición:**

```
GET http://localhost:8080/aulas/1
```

   - **Ejemplo de Respuesta:**

```
{
  "id": 1,
  "nombre": "Aula 101",
  "tamaño": 50,
  "capacidad": 20,
  "foto": null,
  "descripción": "Aula equipada con proyector"
      }
```

3. **GET /api/aulas**
   - **Descripción**: obtiene una lista de todas las aulas disponibles.

   - **Ejemplo de Petición:**

```
GET http://localhost:8080/aulas
```

   - **Ejemplo de Respuesta:**

```
[
  {
    "id": 1,
    "nombre": "Aula 101",
    "tamaño": 50,
    "capacidad": 20,
    "foto": null,
    "descripción": "Aula equipada con proyector"
  },
  {
    "id": 2,
    "nombre": "Aula 102",
    "tamaño": 60,
```

```
  "capacidad": 30,
  "foto": null,
  "descripción": "Aula con sistemas multimedia"
}
    ]
```

4. **DELETE /api /{id}**
   - **Descripción**: elimina un aula por su ID.
   - **Ejemplo de Petición:**

```
DELETE http://localhost:8080/aulas/1
```

   - **Ejemplo de Respuesta:**

```
HTTP/1.1 200 OK "Aula eliminada correctamente"
```

4.1.6 API Autenticación

1. **GET /**
   - **Descripción**: muestra un mensaje de bienvenida al acceder a la aplicación.
   - **Ejemplo de Petición:**

```
GET http://localhost:8080/
```

   - **Ejemplo de Respuesta:**

```
"Welcome to the application!"
```

2. **POST /register**
   - **Descripción**: registra un nuevo usuario.
   - **Cuerpo de la Petición:**

```
{
  "correo": "newuser7@ctb.upm.es",
  "nombre": "Jane",
  "apellido": "Smith",
  "contraseña": "password123",
  "rol": "Subordinado",
  "diasAsuntosPropios": 3,
  "diasVacaciones": 10
      }
```

3. **POST /login**
   - **Descripción**: inicia sesión con las credenciales del usuario (correo y contraseña).
   - **Cuerpo de la Petición:**

```
{
  "correo": "newuser7@ctb.upm.es",
  "contraseña": "password123"
```

```
}
```

- **Ejemplo de Respuesta:**

```
"Login successful. Welcome Jane"
```

4. **POST /logout**
   - **Descripción**: cierra la sesión del usuario actual.
   - **Ejemplo de Petición:**

```
POST http://localhost:8080/logout
```

   - **Ejemplo de Respuesta:**

```
"Logout successful "
```

4.2 Desarrollo de FrontEnd

Para el desarrollo del frontend de la aplicación se utilizó Next.js (véase *Capítulo 2: Tecnologías Empleadas*) y, para facilitar el trabajo en paralelo durante el desarrollo, el frontend se alojó en un repositorio separado del backend. Esto permitió una clara división de responsabilidades y mayor independencia entre ambos módulos.

4.2.1 Organización del Repositorio

El repositorio del frontend está organizado de manera modular, como se observa en la *Figura 9*. Los directorios principales son:

- /app: es el núcleo del frontend y contiene subcarpetas que representan las rutas de la aplicación. Por ejemplo, /reservas gestiona la lógica asociada a la URL https://www.ctb.upm.es/reservas. Dentro de cada carpeta, el archivo page.tsx actúa como punto de entrada, mientras que otros archivos, como CardAula.tsx o AulaList.tsx, encapsulan partes de la lógica en componentes reutilizables. Esto fomenta la separación de responsabilidades y facilita la mantenibilidad del código.

  Además, los archivos types.tsx definen los tipos de datos utilizados en los componentes. Por ejemplo, el tipo Ausencia está definido de la siguiente manera:

```
export interface Ausencia {
  id?: number;
  fechaComienzo: string;
  fechaFin: string;
  tipo: 'Vacaciones' | 'Enfermedad' | 'Asuntos_propios';
```

```
  estado: 'Pendiente' | 'Aceptada' | 'Rechazada';
  investigador: string;
  responsable: string | null;
}
```

- /ui: este subdirectorio contiene componentes de interfaz de usuario, como sidenavadmin.tsx y sidenavusuario.tsx, que son los menús de navegación del administrador y del usuario, respectivamente. Estos menús se invocan desde layout.tsx o page.tsx de cada página, garantizando una navegación consistente en toda la aplicación.

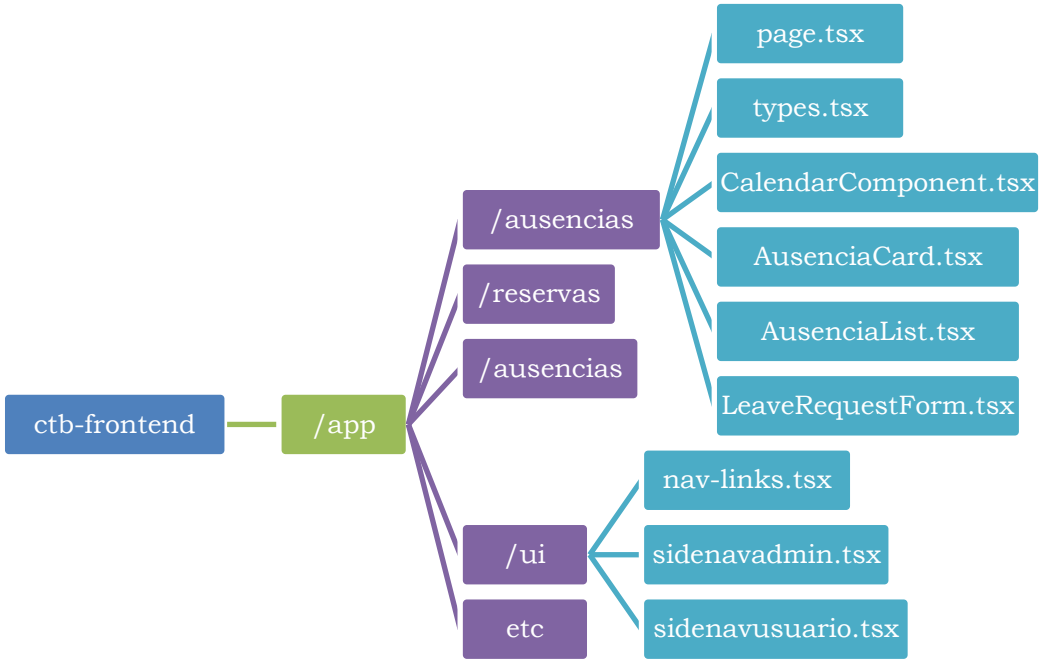

Figura 9: Organización Repositorio Frontend

4.2.2 Personalización de la Interfaz según Roles

Una característica importante de la aplicación es la personalización de las opciones del menú según el rol del usuario, como se observa en la *Figura 10*. Por ejemplo, los usuarios con rol de "Responsable" tienen acceso exclusivo a opciones como "Gestionar Ausencias" y "Disponibilidad de Equipo". Esto se implementó mediante un control en el menú de navegación:

```
{role === "Responsable" && (
  <>
    <li>
      <Link
        href="/gestionar-ausencias"
```

```
                    className="flex items-center gap-4 p-4 rounded-
lg text-lg font-medium text-green-900 border border-green-200
transition duration-300 ease-in-out hover:bg-green-200 hover:-
translate-y-1"
                >
                    <ScaleIcon className="w-6 h-6 text-green-600"
/>
                        <span  className="font-serif">Gestionar
Ausencias</span>
                </Link>
            </li>
```

Este enfoque mejora la usabilidad al ofrecer una experiencia más personalizada y enfocada para cada tipo de usuario.

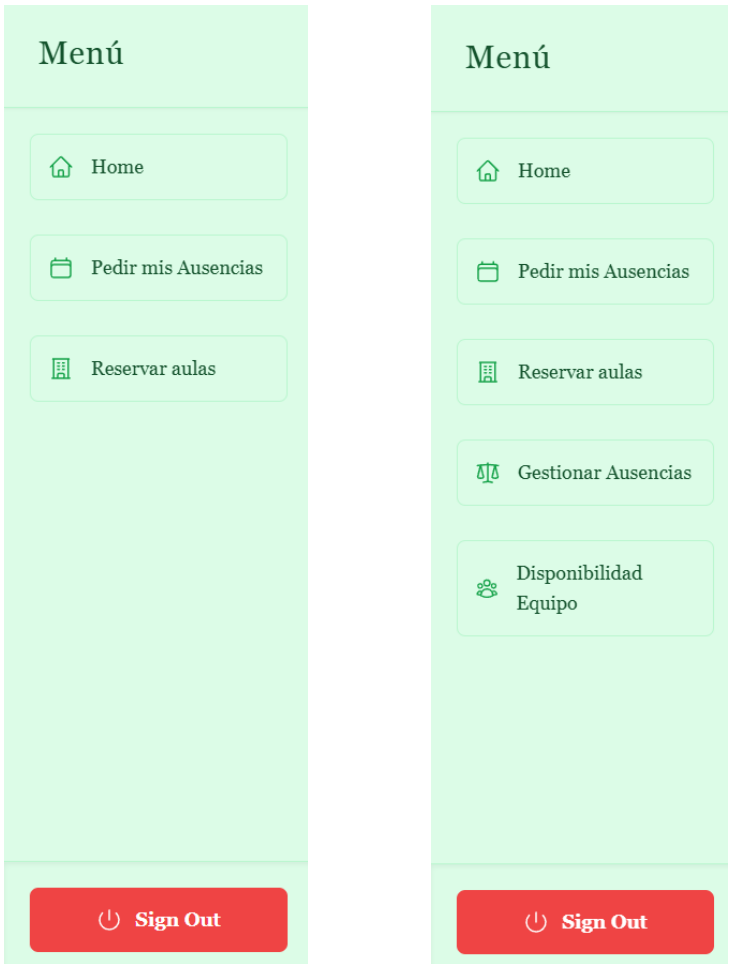

Figura 10: Comparativa de Menús de Subordinado y Responsable

4.2.3 Comunicación Frontend-Backend

El frontend se ejecuta en el puerto 3000, mientras que el backend está alojado en el puerto 8080. La comunicación entre ambos se realiza mediante peticiones API que incluyen los endpoints necesarios. Por ejemplo:

```
    const fetchAusencias = async () => {
      try {
        const userCorreo = localStorage.getItem('email');
        const response = await
fetch(`http://localhost:8080/ausencias/${userCorreo}`, {
          method: 'GET',
          credentials: 'include', // Include cookies for
authentication
          headers: {
            'Content-Type': 'application/json',
          },
        });
```

En este caso, se utiliza localStorage para obtener el correo del usuario autenticado y realizar una solicitud al backend. Cabe destacar que únicamente se almacenan el correo y el rol del usuario en localStorage, siguiendo las buenas prácticas de seguridad para minimizar la exposición de datos sensibles en caso de un ataque.

4.2.4 Configuración de CORS en el Backend

Dado que el frontend y el backend operan en diferentes puertos, fue necesario habilitar CORS (Cross-Origin Resource Sharing) en el backend. Esto se logró añadiendo la siguiente configuración en Spring Boot:

```
@Bean
public WebMvcConfigurer corsConfigurer() {
    return new WebMvcConfigurer() {
        @Override
        public void addCorsMappings(CorsRegistry registry) {
            registry.addMapping("/**")
                    .allowedOrigins("http://localhost:3000")
                    .allowedMethods("GET", "POST", "PUT",
"DELETE", "PATCH")
                    .allowedHeaders("*")
                    .allowCredentials(true);
        }
    };
}
```

Esta configuración permite que el frontend realice peticiones al backend mientras mantiene controles de seguridad, como restringir las solicitudes a métodos y orígenes específicos.

4.3 Testeo de la Aplicación

Para garantizar la correcta funcionalidad de todos los servicios y endpoints en el backend, se implementaron clases de pruebas unitarias que pueden ejecutarse de manera individual o colectiva mediante el comando `mvn test`.

Se utilizó Mockito para simular el comportamiento de los servicios y repositorios que interactúan con las clases bajo prueba. Esto permite evaluar la lógica de negocio de forma aislada y asegura que las dependencias no interfieran en los resultados de las pruebas.

La estructura general de las pruebas sigue el patrón Arrange-Act-Assert:

- Arrange: se prepara el entorno de prueba y se configuran las simulaciones de las dependencias.
- Act: se ejecuta el método que se desea probar.
- Assert: se verifican los resultados esperados mediante aserciones, como validar el código de estado HTTP, mensajes de respuesta o invocaciones a servicios simulados.

A continuación, se presenta una prueba unitaria para la funcionalidad de eliminar un aula en el Controlador:

```
@Test
public void testDeleteAula() {
    // Arrange: Preparar los datos y simular el comportamiento
del servicio
    Integer aulaId = 1;
    doNothing().when(aulaService).deleteAulaById(aulaId);

    // Act: Ejecutar el método del controlador

    ResponseEntity<String> response =
aulaController.deleteAula(aulaId);

    // Assert: Verificar los resultados esperados

    assertEquals(HttpStatus.OK, response.getStatusCode(), "El
código de estado debería ser 200 OK");
    assertEquals("Aula eliminada correctamente",
response.getBody(), "El mensaje de respuesta no coincide");
```

```
    verify(aulaService, times(1)).deleteAulaById(aulaId);


// Confirmar que el servicio fue llamado una vez
}
```

Explicación del Ejemplo:

doNothing(): se configura el servicio simulado (aulaService) para que no realice ninguna operación al invocar el método deleteAulaById.

ResponseEntity: representa la respuesta del controlador. Aquí se verifican tanto el código de estado HTTP (HttpStatus.OK) como el mensaje del cuerpo de la respuesta.

verify: Se asegura que el método deleteAulaById fue invocado exactamente una vez con el ID del aula.

A continuación, se detalla una prueba unitaria que valida la funcionalidad de eliminar un aula por ID en la capa de servicio:

```
  @Test
    public void testDeleteAulaById() {
        // Arrange

when(aulaRepository.findById(1)).thenReturn(Optional.of(aula1));
        doNothing().when(aulaRepository).delete(aula1);


        // Act
        aulaService.deleteAulaById(1);


        // Assert
        verify(aulaRepository, times(1)).delete(aula1);
    }
```

Explicación del Ejemplo:

- when(aulaRepository.findById(1)).thenReturn(Optional.of(aula1)): simula que, al buscar un aula con ID 1, el repositorio devuelve un objeto aula1 envuelto en un Optional. Esto representa un aula existente.

- doNothing().when(aulaRepository).delete(aula1): indica que, al invocar el método delete del repositorio, no se realizará ninguna acción. Esto se usa para evitar dependencias externas en el test.

- aulaService.deleteAulaById(1): se ejecuta el método del servicio que está siendo probado. Este método debería:
    - Verificar si el aula existe.
    - Eliminarla si se encuentra disponible.
- verify(aulaRepository, times(1)).findById(1): comprueba que el repositorio buscó el aula por ID exactamente una vez.
- verify(aulaRepository, times(1)).delete(aula1): verifica que el repositorio eliminó el aula encontrada.

Además de las pruebas unitarias implementadas, se han llevado a cabo pruebas funcionales utilizando Postman. Estas pruebas consistieron en enviar solicitudes HTTP a los endpoints correspondientes y verificar que:

1. Las respuestas obtenidas son las esperadas:
    - Los códigos de estado (como 200 OK, 404 Not Found o 400 Bad Request) corresponden a los escenarios probados.
    - Los cuerpos de las respuestas contienen los datos adecuados o los mensajes de error correctos.
2. La base de datos se actualiza correctamente:
    - Se confirmó que las operaciones CRUD (crear, leer, actualizar y eliminar) aplicadas mediante los endpoints producen los cambios previstos en la base de datos.

Para ver ejemplos de lo dicho, consúltese el *Apartado 6.2 del Anexo*.

Finalmente, una vez completado el desarrollo del frontend, se llevaron a cabo pruebas exhaustivas para garantizar el correcto funcionamiento de todos los componentes y la integración con la base de datos. Estas pruebas se realizaron en tiempo real, observando cómo los datos eran procesados y almacenados en el sistema. Por ejemplo, se verificó que al solicitar una ausencia de tipo "Vacaciones" con una fecha de inicio posterior a la fecha de fin, el sistema detectara y rechazara correctamente este error. Este tipo de validación también se probó con otros tipos de ausencias, como "Asuntos Propios" y "Enfermedad", asegurando la consistencia en las reglas de negocio implementadas.

Adicionalmente, se llevaron a cabo pruebas de casos de uso correctos para comprobar que las solicitudes realizadas de manera adecuada se procesaban correctamente. Esto incluyó validar que, tras realizar una petición, la base de datos se actualizaba como era esperado y, en el caso del ejemplo de pedir una ausencia, que el responsable asignado podía visualizar la solicitud correspondiente en su sección de “Gestionar Ausencias”. De esta manera, se confirmó que el sistema cumplía con los flujos establecidos.

Otro aspecto importante que se verificó fue el envío de notificaciones automáticas. Para ello, se revisaron las bandejas de correo asociadas a los usuarios involucrados, comprobando que las notificaciones llegaban correctamente y en los momentos esperados.

Gracias a estas pruebas exhaustivas, se pudo garantizar que el sistema funciona conforme a los requisitos establecidos y que ofrece una experiencia confiable para los usuarios.

5 Resultados y conclusiones

El presente Trabajo de Fin de Grado (TFG) ha tenido como objetivo principal el diseño y desarrollo de una plataforma web para optimizar la gestión de recursos en el Centro de Tecnología Biomédica (CTB). Este objetivo se planteó como respuesta a problemáticas específicas identificadas, tales como la falta de trazabilidad, la duplicación de datos y la ausencia de un sistema centralizado para la gestión de reservas de aulas, así como del control de ausencias y vacaciones del personal. La solución propuesta aborda estas limitaciones mediante una herramienta escalable, eficiente y adaptada a las demandas del entorno del CTB, cumpliendo con los objetivos generales y específicos establecidos al inicio del proyecto.

En primer lugar, se realizó un análisis exhaustivo de las necesidades del centro, que permitió seleccionar las tecnologías más adecuadas para garantizar el rendimiento, la seguridad y la escalabilidad del sistema. Estas decisiones sentaron una base sólida para el desarrollo de la plataforma, asegurando su capacidad de soportar futuras ampliaciones y mantener altos estándares de calidad. Además, se diseñó una arquitectura basada en principios de ingeniería de software, como la responsabilidad mínima y la independencia de las clases, lo que garantizó la creación de un sistema modular, mantenible y adaptable a nuevas funcionalidades.

Durante la implementación, se desarrollaron dos módulos principales. El primero, enfocado en la gestión de aulas, permite realizar reservas de manera eficiente y evita conflictos mediante validaciones automáticas (es decir, si un usuario reserva un espacio, la página no permite al resto de usuarios reservarlo en ese intervalo). El segundo, centrado en la gestión del personal, facilita la administración de ausencias y vacaciones, reduciendo superposiciones y proporcionando transparencia en la planificación. Estos módulos fueron diseñados sobre una base de datos escalable y segura, que actúa como núcleo del sistema, y una interfaz de usuario intuitiva que asegura una experiencia accesible para todos los perfiles del centro.

Para garantizar la seguridad del sistema, se implementaron mecanismos de cifrado para los datos sensibles y un control de acceso basado en roles, protegiendo tanto la privacidad de la información como la integridad de las operaciones. Asimismo, se realizaron pruebas exhaustivas, incluyendo testeo unitario, para validar el cumplimiento de los requisitos establecidos y asegurar que la plataforma satisficiera las necesidades del CTB.

A lo largo del proyecto, se han adquirido conocimientos y habilidades clave en diferentes áreas. En el ámbito técnico, se han profundizado los conocimientos sobre diseño de bases de datos, desarrollo de interfaces de usuario y aplicación de buenas prácticas en la arquitectura del software. Además, se ha aprendido a gestionar de manera eficiente un proyecto de software, desde la recopilación de requisitos hasta la fase de pruebas, pasando por el diseño y la implementación. También se ha adquirido experiencia en la resolución de problemas prácticos y en la toma de decisiones estratégicas, teniendo siempre en cuenta el impacto del sistema en términos de escalabilidad, seguridad y usabilidad.

Más allá de los aspectos técnicos, este proyecto ha permitido comprender mejor la importancia de considerar el impacto social, económico y medioambiental de

las soluciones tecnológicas. Por ejemplo, la implementación de un sistema que optimiza la gestión de recursos puede reducir significativamente los costos operativos y liberar tiempo para actividades más valiosas, como la investigación.

En conclusión, este TFG no solo ha dado lugar a una solución funcional y escalable para el CTB, sino que también ha supuesto un valioso proceso de aprendizaje, combinando habilidades técnicas, organizativas y analíticas. Además, deja abierta la puerta a futuros desarrollos que expandan el alcance del sistema y sigan contribuyendo a la mejora de los procesos administrativos del centro (véase el *Capítulo 6*).

6 Análisis de Impacto

A lo largo del desarrollo de este Trabajo de Fin de Grado (TFG), se han analizado los impactos potenciales de los resultados obtenidos en diversos contextos, incluyendo el personal, empresarial, social, económico, medioambiental y cultural. La solución propuesta no solo aborda problemas específicos del Centro de Tecnología Biomédica (CTB), como la gestión de aulas y ausencias, sino que también sienta las bases para una mayor escalabilidad, permitiendo en el futuro gestionar otros recursos del centro de manera eficiente y sostenible.

El proyecto contribuye directamente al cumplimiento de los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030, especialmente al Objetivo 9: Industria, Innovación e Infraestructura. En particular, se alinea con la meta 9.5, que busca fomentar la innovación y fortalecer las capacidades tecnológicas de los sectores industriales a través de la investigación y el desarrollo. La plataforma desarrollada mejora la trazabilidad y la organización interna del centro, promoviendo un uso más eficiente de los recursos y un entorno de trabajo más colaborativo, lo que incrementa su potencial para generar innovación. Además, la aplicación reduce la carga burocrática del personal permitiéndoles centrarse más en sus tareas de investigación.

Asimismo, se ha tenido en cuenta el impacto medioambiental del sistema, priorizando principios de diseño sostenibles y promoviendo la reducción del uso de recursos físicos, como papel, mediante la digitalización de procesos administrativos. Además, la solución garantiza un acceso inclusivo y equitativo, mejorando la experiencia de usuario para todos los perfiles del centro y fomentando una mayor transparencia en la gestión.

En términos personales y empresariales, este trabajo representa una contribución significativa al fortalecimiento de las competencias técnicas y metodológicas necesarias para abordar problemas reales, al mismo tiempo que permite al centro optimizar sus procesos y liberar recursos humanos para actividades de investigación. En el plano económico, se espera una reducción de costes operativos gracias a la mejora en la eficiencia, mientras que en el ámbito social, el sistema impulsa la cohesión y coordinación entre los miembros del equipo.

Como líneas de desarrollo futuro, se sugiere ampliar la funcionalidad del sistema para incluir la gestión de otros recursos del centro, como equipos tecnológicos, laboratorios y proyectos de investigación. Esto permitiría crear una plataforma integral que abarque todas las necesidades administrativas del CTB. Otro aspecto a considerar es la integración con sistemas externos, como calendarios corporativos, con el objetivo de mejorar la interoperabilidad y la eficiencia. Además, podría automatizarse el proceso de gestión de los festivos mediante la consulta del calendario oficial de la Comunidad de Madrid. Por último, se recomienda explorar la posibilidad de incorporar tecnologías avanzadas, como la inteligencia artificial, para optimizar tareas recurrentes y proporcionar recomendaciones automatizadas en la planificación de recursos.

En conclusión, los resultados obtenidos no solo resuelven problemas concretos del CTB, sino que también ofrecen una solución con un impacto amplio y sostenible, capaz de adaptarse a las necesidades futuras del centro y de contribuir al desarrollo global en múltiples dimensiones.

.

Bibliografía

[1] Gackenheimer, C. (2015). *Introduction to React*. Apress.

[2] Thakkar, M., & Thakkar, M. (2020). Next. js. *Building React Apps with Server-Side Rendering: Use React, Redux, and Next to Build Full Server-Side Rendering Applications*, 93-137.

[3] Saks, E. (2019). JavaScript Frameworks: Angular vs React vs Vue.

[4] Mythily, M., Raj, A. S. A., & Joseph, I. T. (2022, July). An analysis of the significance of spring boot in the market. In *2022 International Conference on Inventive Computation Technologies (ICICT)* (pp. 1277-1281). IEEE.

[5] Forcier, J., Bissex, P., & Chun, W. J. (2008). *Python web development with Django*. Addison-Wesley Professional.

[6] Tilkov, S., & Vinoski, S. (2010). Node. js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, *14*(6), 80-83.

[7] Yang, D. (2010). *Java persistence with JPA*. Outskirts Press.

[8] Christudas, B., & Christudas, B. (2019). *MySQL* (pp. 877-884). Apress.

[9] Chauhan, A. (2019). A review on various aspects of MongoDB databases. *International Journal of Engineering Research & Technology (IJERT)*, *8*(05), 90-92.

[10] Ordóñez, M. P. Z., Ríos, J. R. M., & Castillo, F. F. R. (2017). *Administración de Bases de datos con PostgreSQL* (Vol. 19). 3Ciencias.

[11] IEEE. "IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications." Octubre de 1998.

[12] IBM. (2024, May 22). *Explicación de las bases de datos relacionales* | IBM. Ibm.com. https://www.ibm.com/es-es/topics/relational-databases, consultado en octubre de 2024

[13] Sharif, A. (2022, December 21). *What Is CRUD? Create, Read, Update, and Delete* | CrowdStrike. Crowdstrike.com.

https://www.crowdstrike.com/cybersecurity-101/observability/crud/, consultado en octubre de 2024

[14] IBM. (2023). *What is Three-Tier Architecture* | IBM. Www.ibm.com. https://www.ibm.com/topics/three-tier-architecture, consultado en noviembre de 2024

Anexo A

A.1 Figma: Capturas del prototipo de alta fidelidad (HiFi)

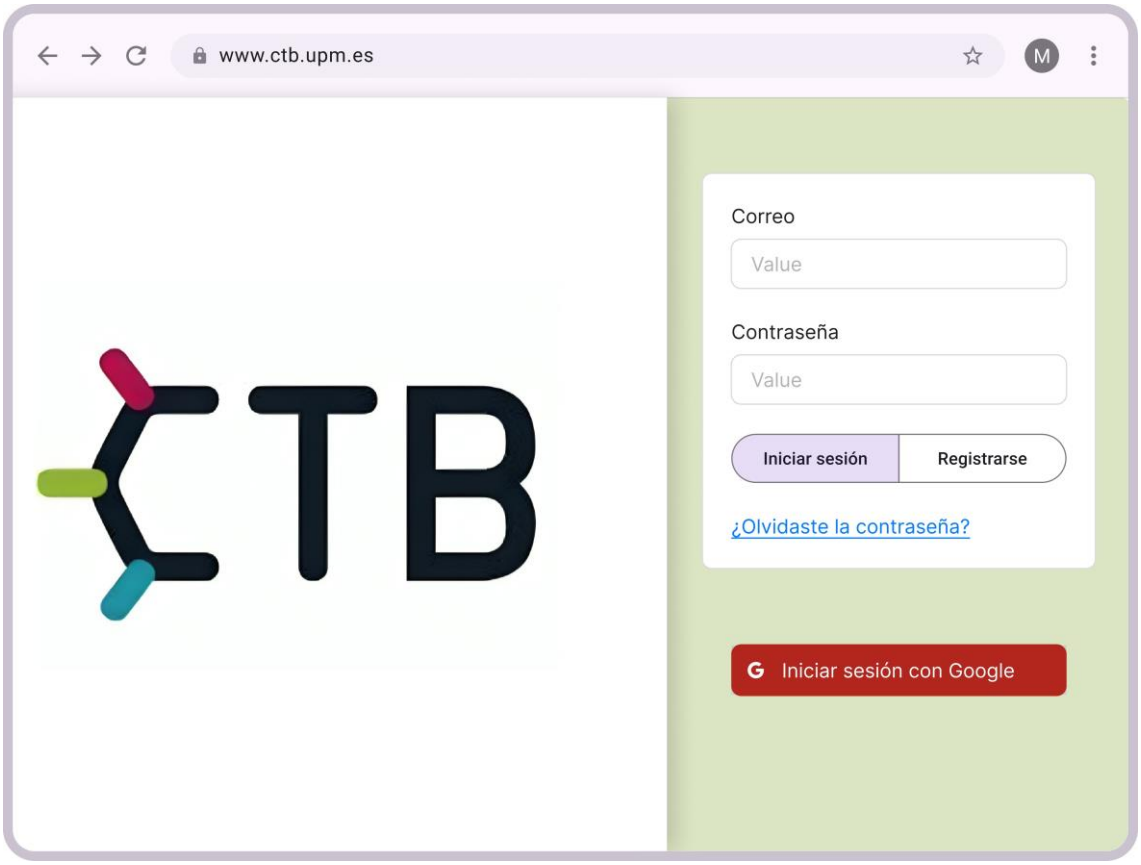

Figura 11: Pantalla de Iniciar Sesión

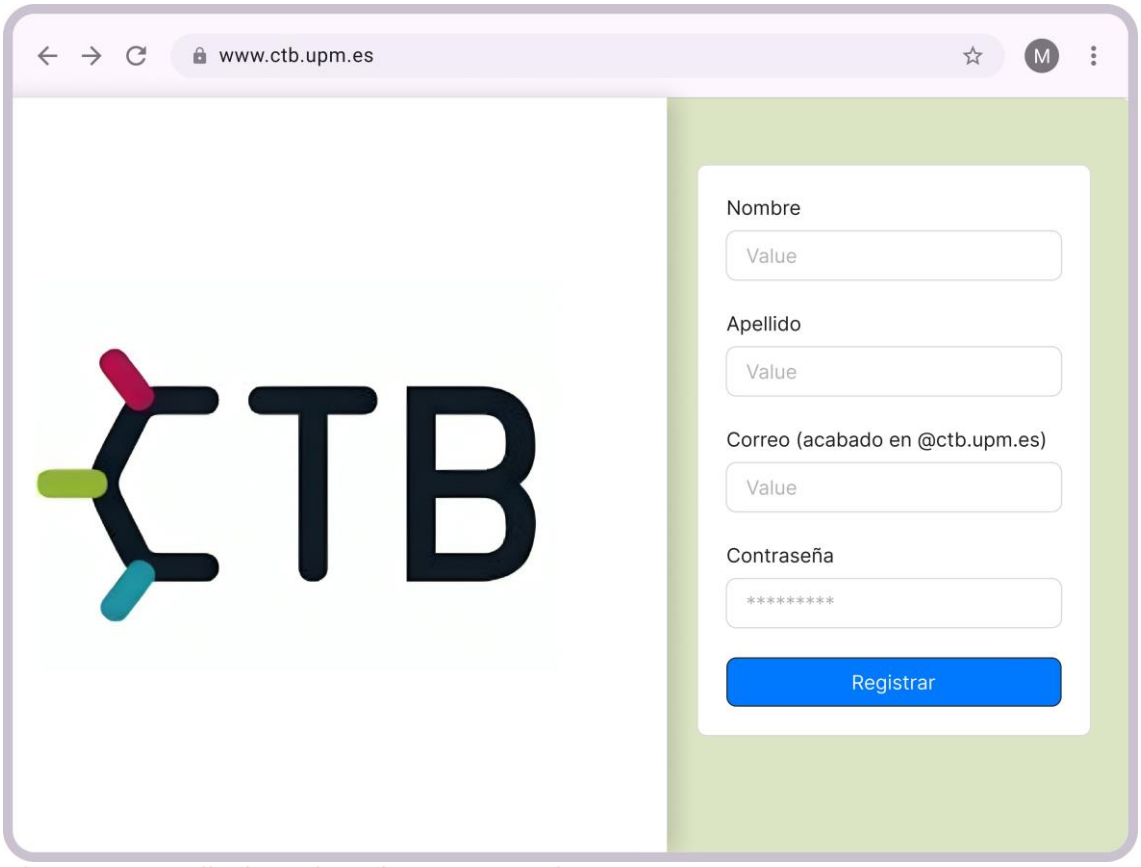

Figura 12: Pantalla de Registro de Nuevo Usuario

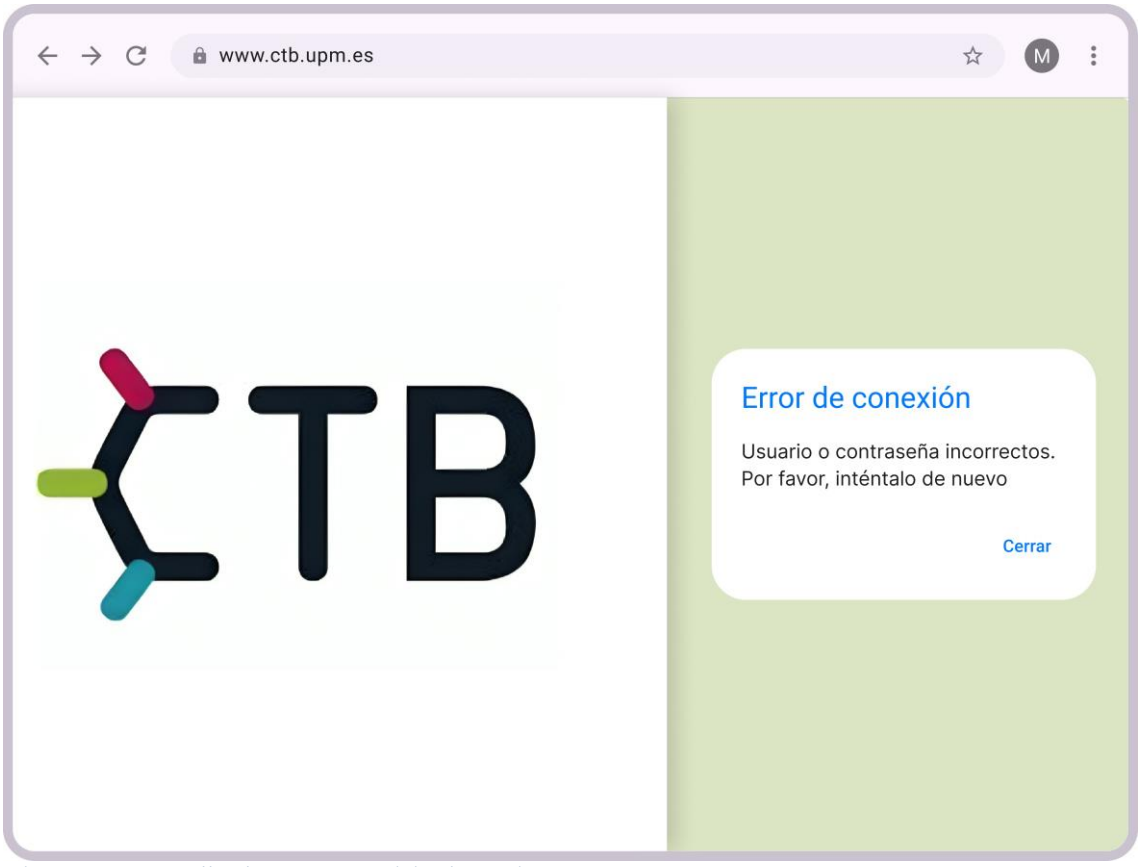

Figura 13: Pantalla de Error en Inicio de Sesión 1

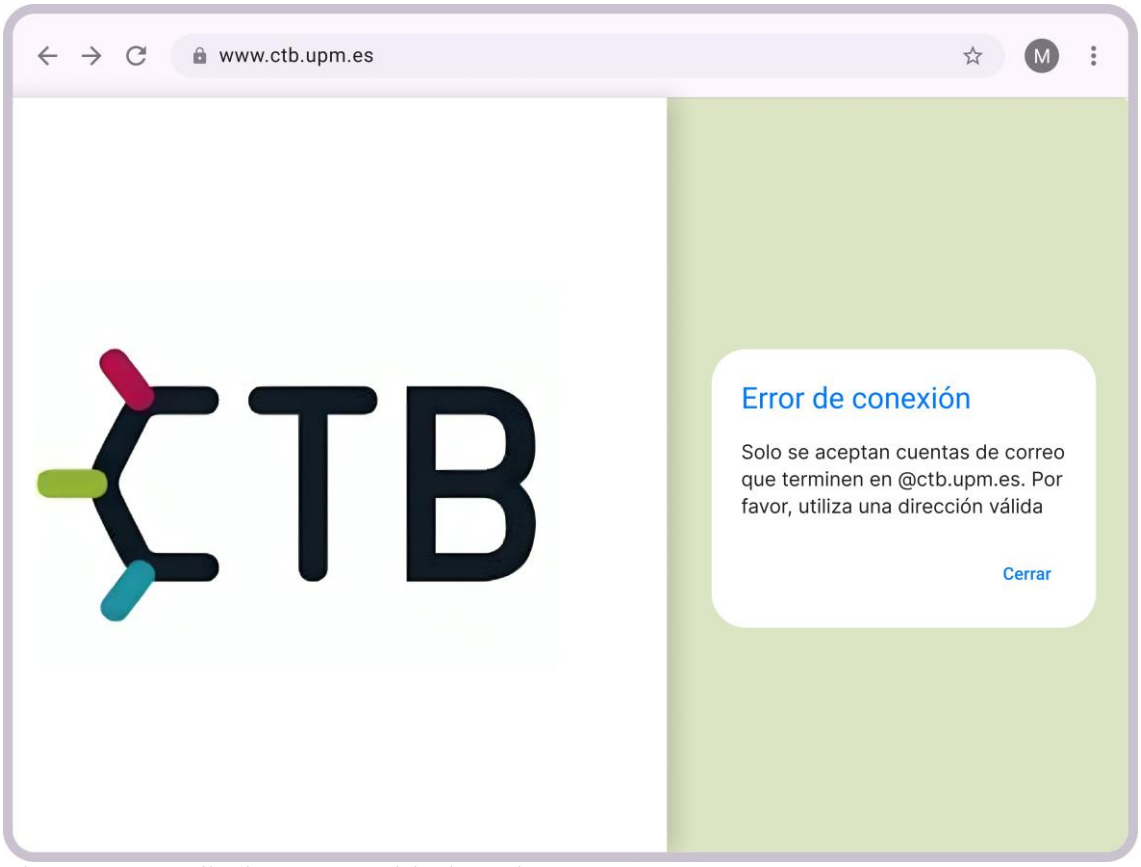

Figura 14: Pantalla de Error en Inicio de Sesión 2

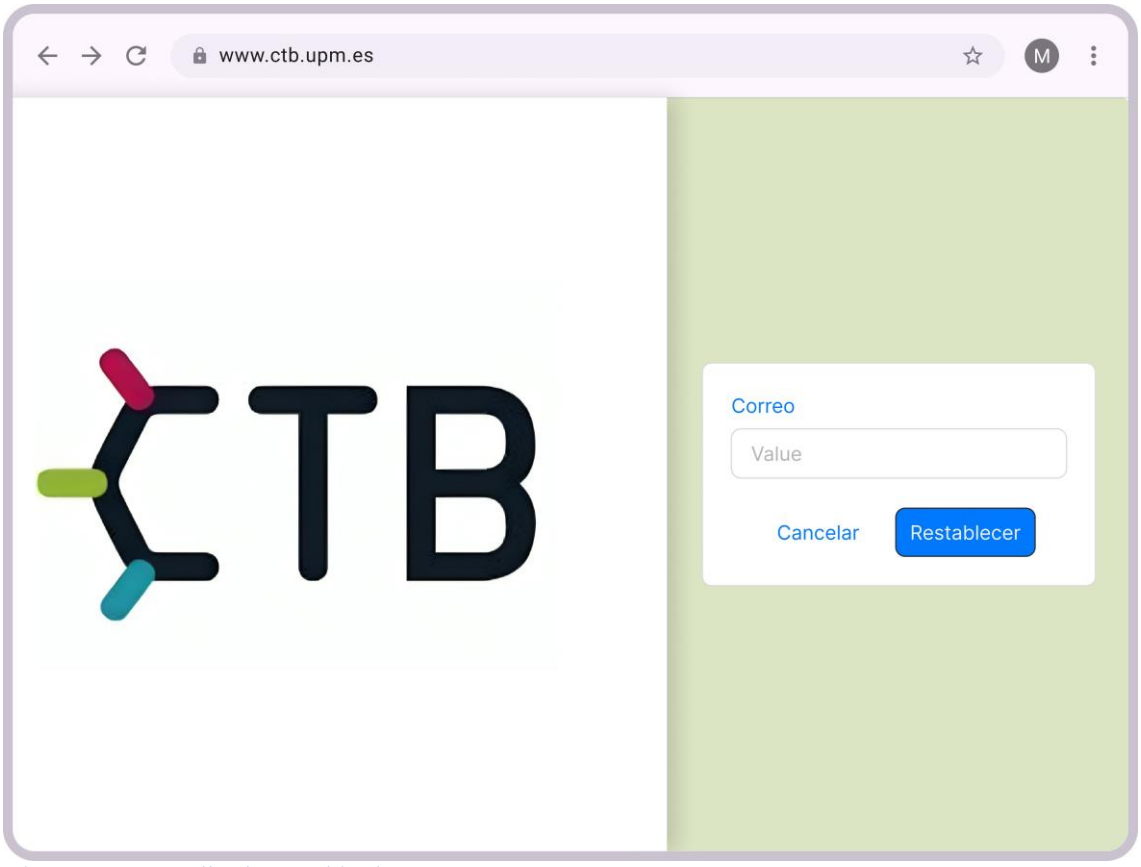

Figura 15: Pantalla de Cambio de Constraseña

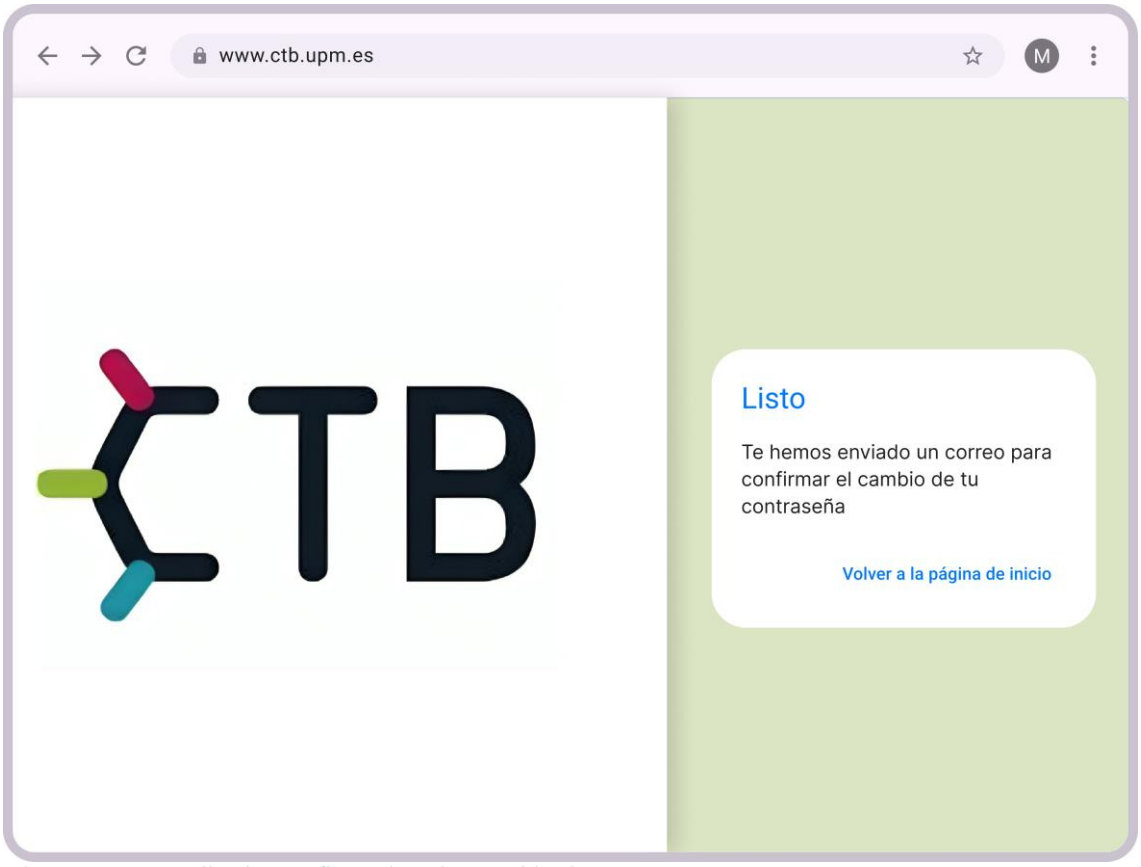

Figura 16: Pantalla de Confirmación de Cambio de Contraseña

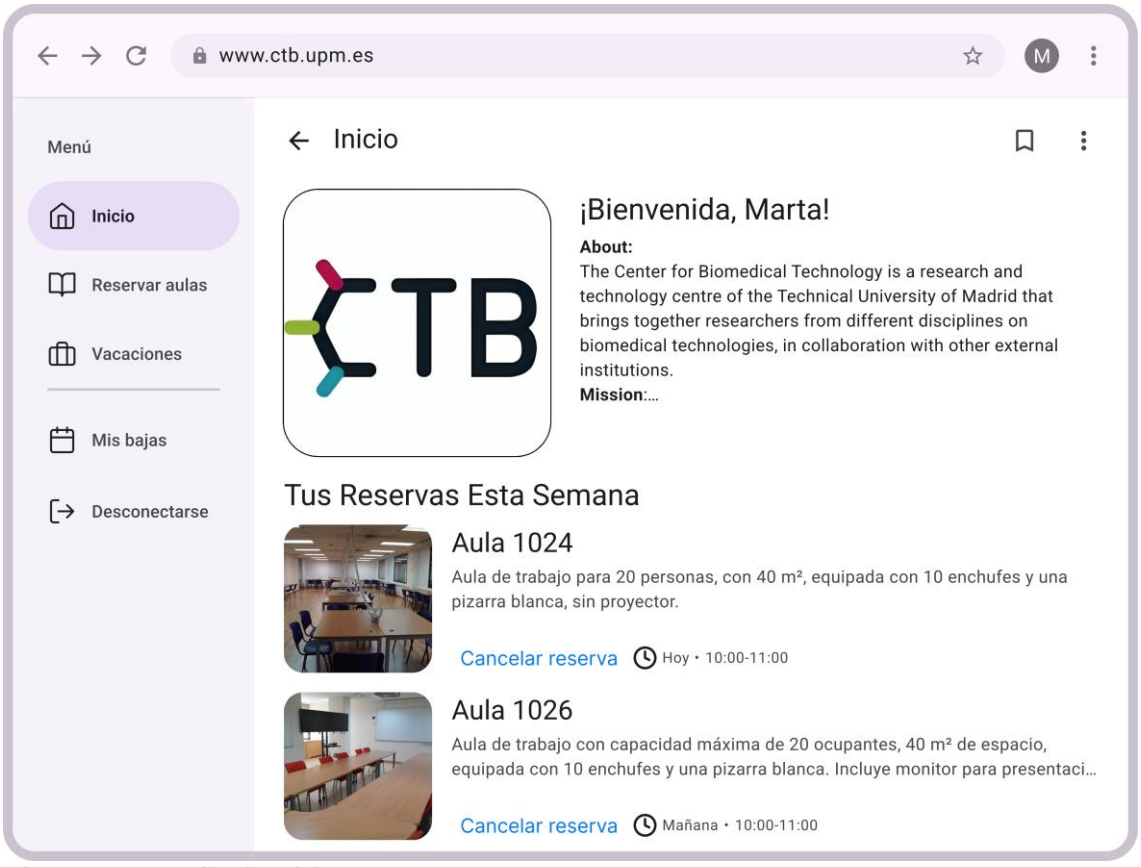

Figura 17: Pantalla de Inicio

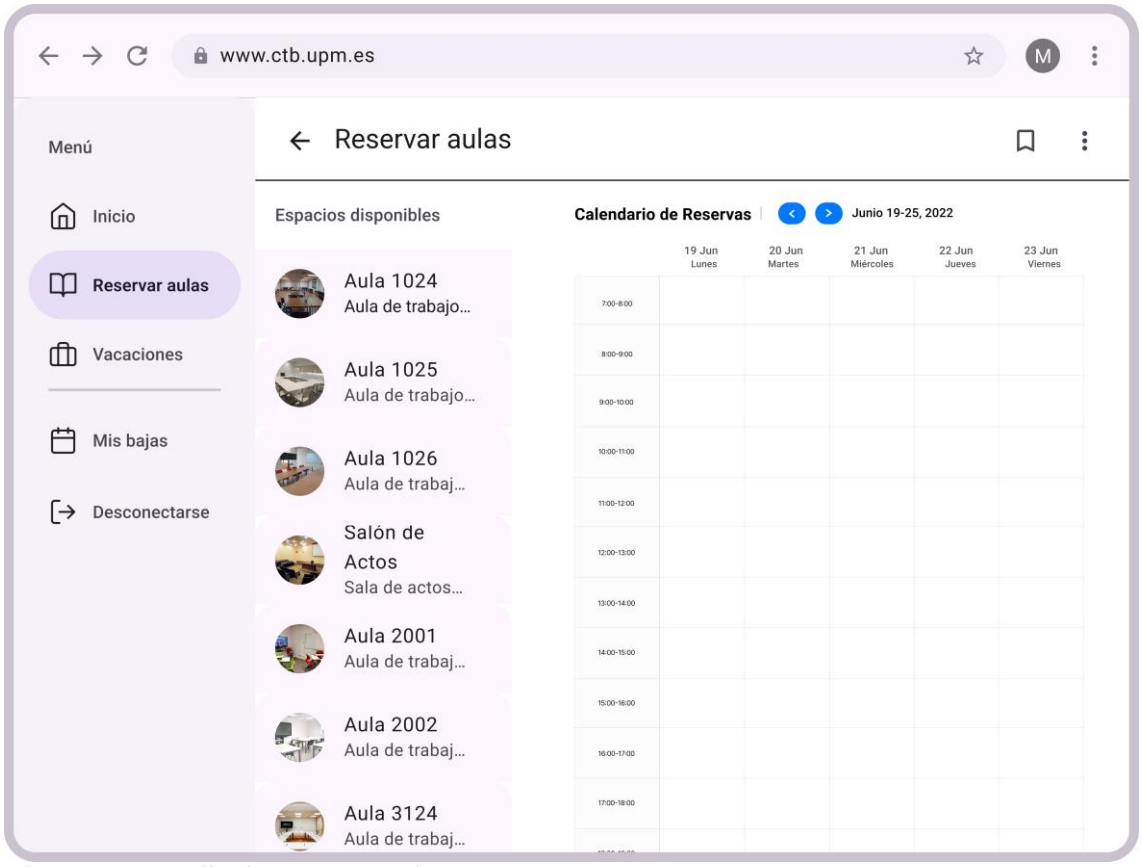

Figura 18: Pantalla de Reservar Aulas

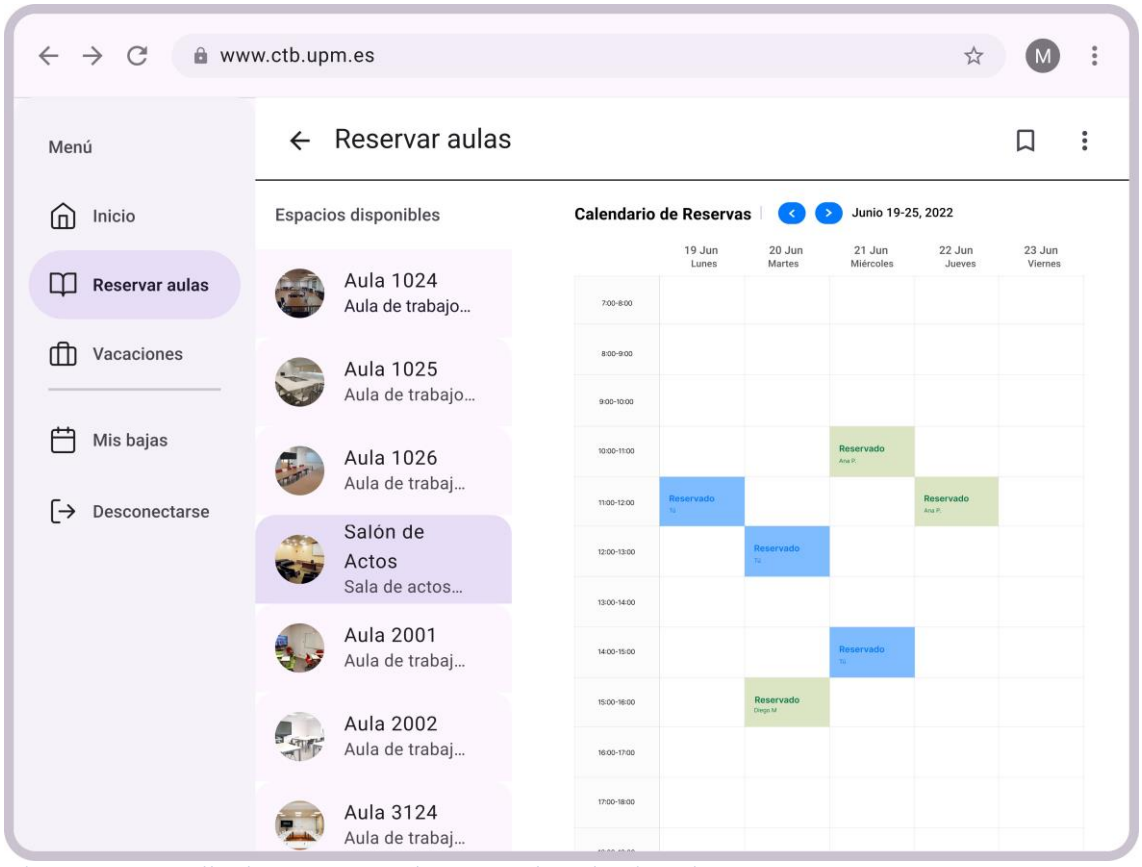

Figura 19: Pantalla de Reservar Aulas con Aula Seleccionada

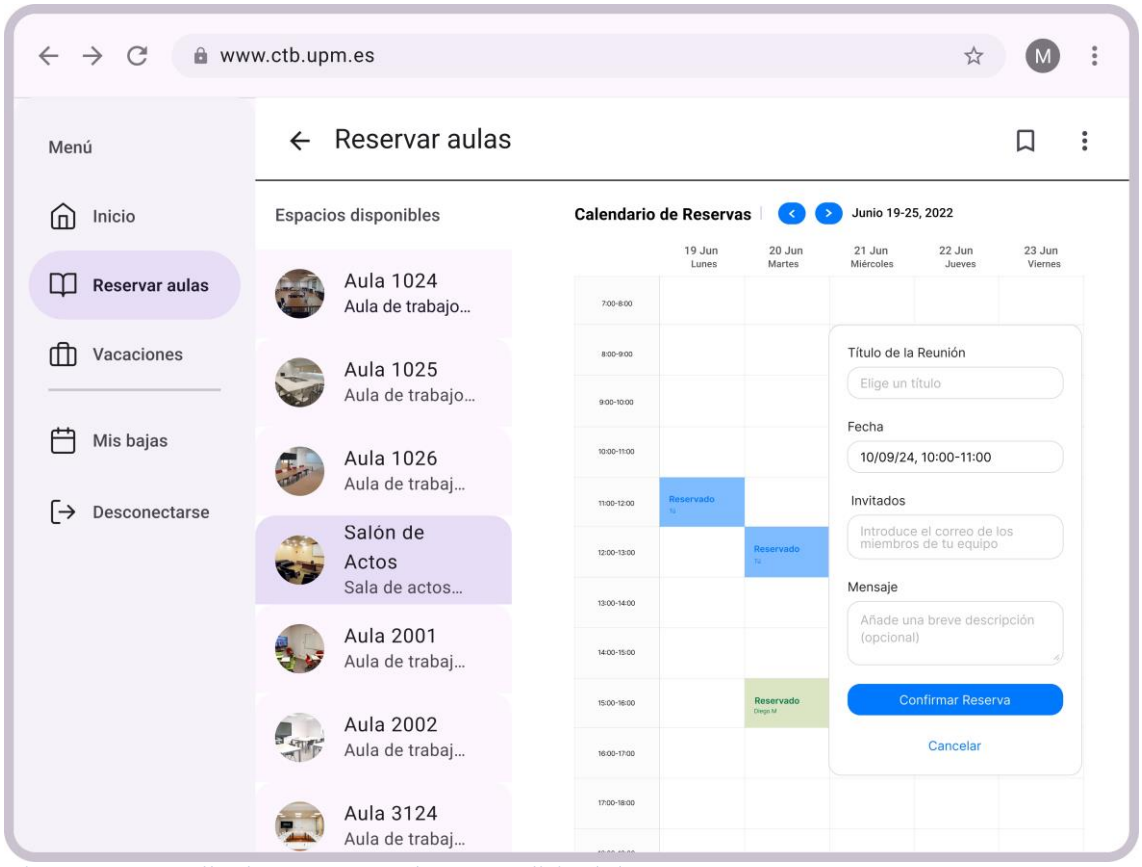

Figura 20: Pantalla de Reservar Aulas con Solicitud de Reserva

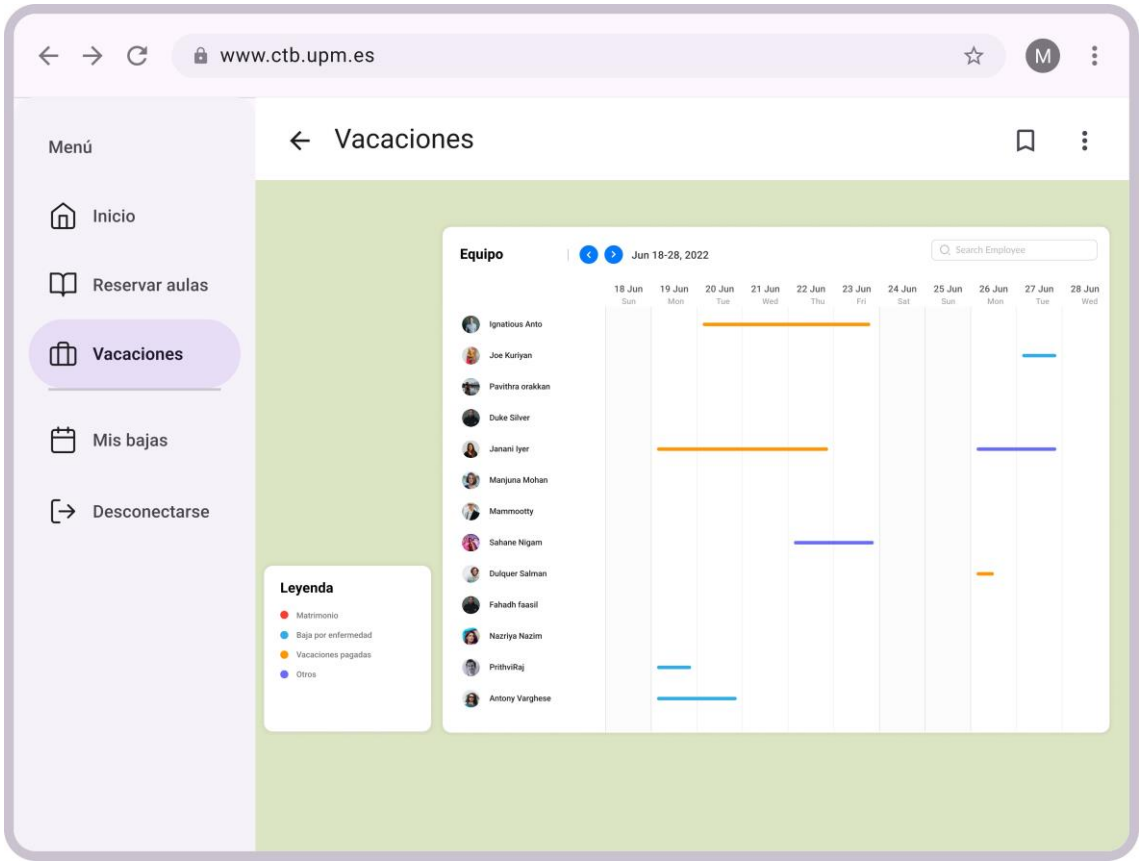

Figura 21: Pantalla de Vacaciones del Equipo

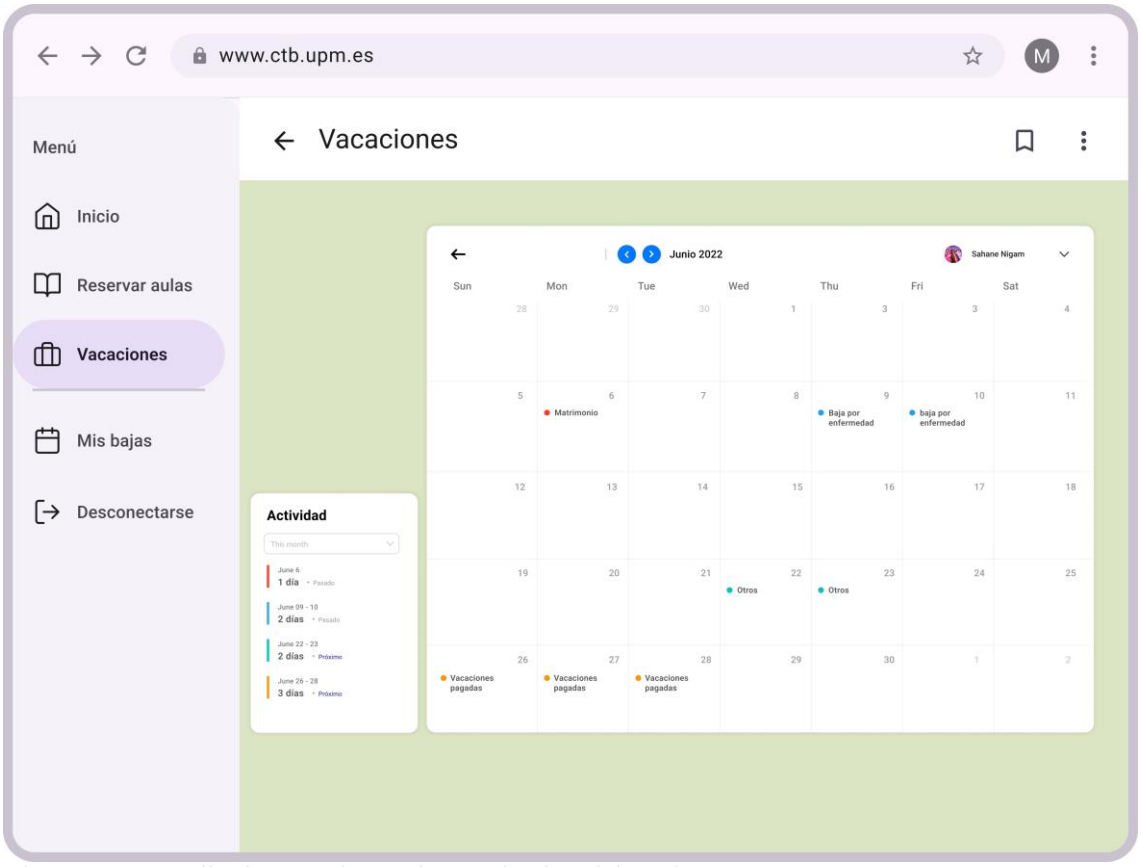

Figura 22: Pantalla de Vacaciones de un Miembro del Equipo

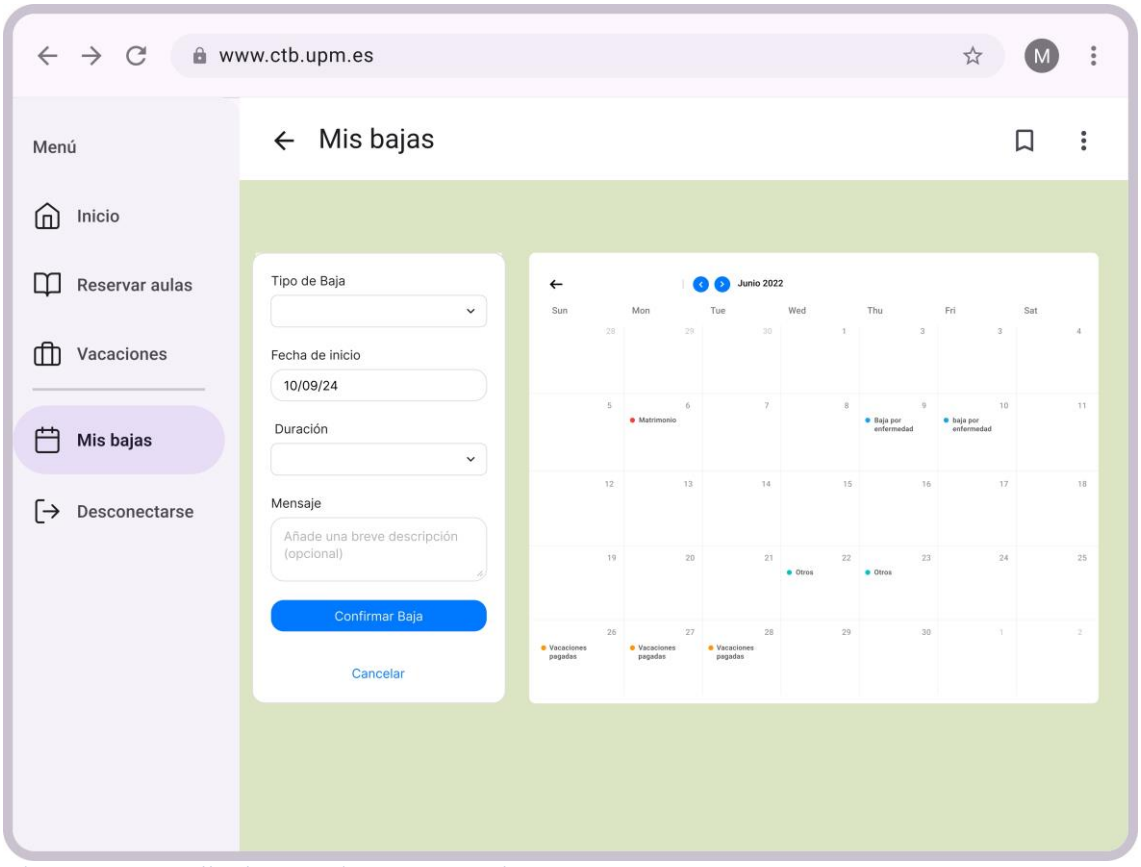

Figura 23: Pantalla de Vacaciones Personales

A.2 Postman: Capturas de Tests de Integración

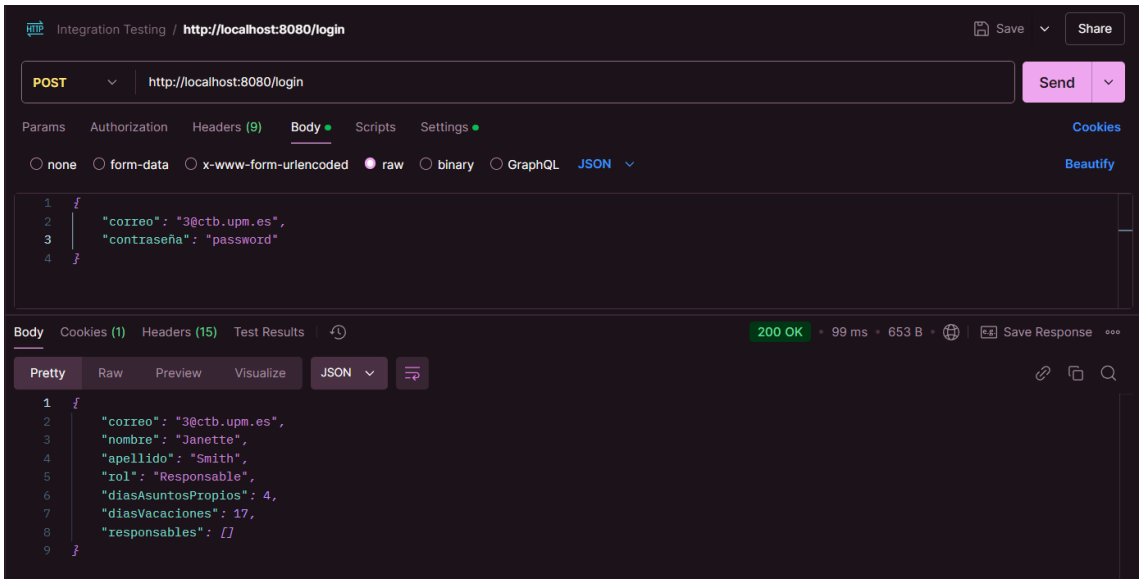

Figura 24: Test Integración en Postman I

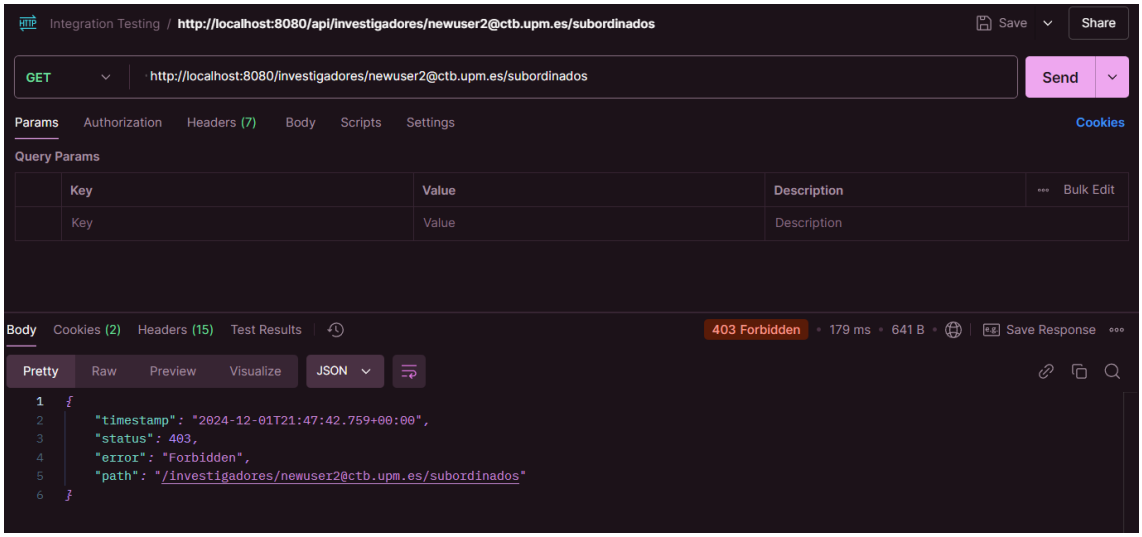

Figura 25: Test Integración en Postman II

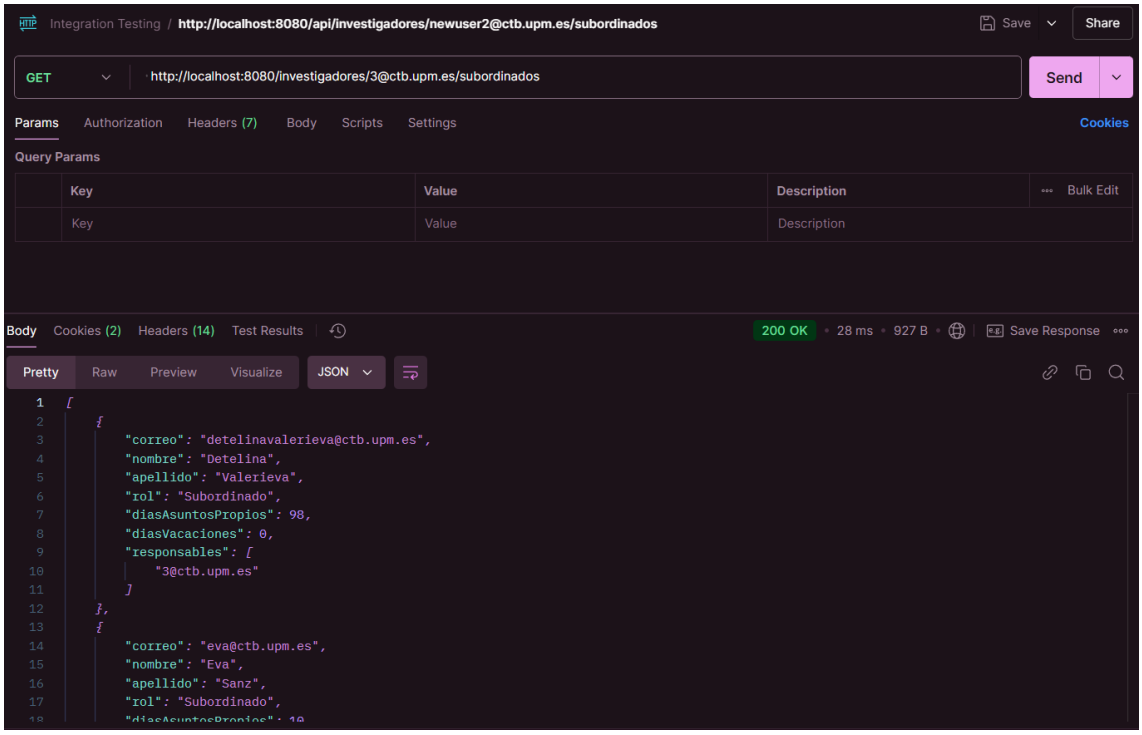

Figura 26: Test Integración en Postman III